DOCTOR OF ECONOMICS AND BUSINESS MANAGEMENT

Essays on Requirements Optimization Problems in the Core Ontology for Requirements Engineering

Gillain, Joseph

Award date:
2017

Awarding institution:
University of Namur

University of Namur
Faculty of Economics, Social Sciences and Business Administration
Department of Business Administration

ESSAYS ON REQUIREMENTS OPTIMIZATION PROBLEMS IN THE CORE ONTOLOGY FOR REQUIREMENTS ENGINEERING

Thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in Management Sciences

Joseph Gillain

June 2017

Supervisors
Prof. Stéphane Faulkner (UNamur)
Prof. Monique Snoeck (KULeuven)

Jury
Dr. Ivan J. Jureta (UNamur)
Prof. Yves Wautelet (KULeuven)
Prof. Nauman A. Qureshi (KFU)
President
Prof. Annick Castiaux (UNamur)

Graphisme de couverture : ©Presses Universitaires de Namur

Rempart de la Vierge, 13
B-5000 Namur (Belgique)

Imprimé en Belgique
ISBN: 978-2-87037-993-6
Dépôt légal: D/2017/1881/32
Impression Université de Namur

Abstract

In contemporary societies, Information Systems have become ubiquitous. We use them, for instance, in accounting, marketing, human resource and also beyond the business world in health care, education or even daily life activities. It has become common to say that such systems should be of high quality, that is, they should ensure the satisfaction of the needs they were intended for.

Requirements Engineering is the field dedicated to investigate those requirements as well as the conditions for their satisfaction. However, previous and current research has mainly concentrated its effort on methods and concepts able to prove that a particular system could fulfill the requirements. In contrast, few effort had the purpose of discussing about how we should compare alternative solutions and select the optimal one.

This thesis suggests a new formulation of the Requirement Problem which sets the process of optimization in the center of concerns. It shows how different criteria can be used to design optimization models and provides tools to resolve them. Through those models, we describe how the fields of Requirement Engineering and Mathematical Optimization can be connected. The present work divides into four parts.

Part I introduces the problem and outlines the overall context. A case, which is used as example through this thesis, is also presented. Then, we lay the foundations for deeper considerations by describing basic theoretical concepts. This first part is concluded with a discussion about the underlying hypothesis and limitations.

Part II presents four distinct Requirements Problems that we formalize as Optimization Problems. We describe how to model them with Techne, a Requirement Modeling Language, and show how to map them to a mathematical model. The first model addresses the issue of providing an optimal planning. Secondly, we present the problem of finding the optimal reusability level when developing information systems for different customers. The third model focuses on finding the minimal cost between alternative solutions. Eventually, we present self-configurable systems, i.e. systems that provide users with configuration capabilities, and suggest a trade-off model to determine the right amount of configurability those systems should have.

Part III presents AnalyticGraph, the tool we developed to support our contributions. AnalyticGraph proves the feasibility of the discussed optimization models and extends current RE software capabilities.

Part IV sketches possible further research directions on Requirements Optimization and concludes the present work.

Acknowledgments

I thank my thesis supervisor Stéphane Faulkner for guiding and supporting me over the years. He sets the environment necessary to complete this work. His guidance helped me all the time of research and writing.

I also would like to thank my co-supervisor, Monique Snoeck, her advices have been particularly beneficial and her availability was very precious.

I thank Ivan Jureta without whom this work would have not achieved the current maturity. His availability, his straightforwardness and his inestimable advices helped me to elaborate and refine my ideas into more rigorous statements.

I thank the other members of the jury, Yves Wautelet and Nauman Qureshi, for their very valuable feedback that help me to improve the final version of this thesis.

I thank my colleagues and the members of the faculty. Their suggestions were very inestimable. In particular, I would like to thank Corentin Burnay for his collaboration regarding the design, implementation and test of AnalyticGraph.com.

I could not finish these acknowledgments without thanking my family and my close friends. They all were incredibly supportive. Especially Sarah, whose patience, I have to admit, was extremely challenged several times but she never stopped support me.

Finally, I dedicate this work to my mother who, I am sure of that, would have been much more proud of this achievement that I will ever be. Thank you.

Contents

Part I

Research Context

1 Introduction

1.1 Research Context: Requirement Engineering

Information Systems (ISs) are any organized system for the collection, organization, storage and communication of information. Since the mid 20^{th} century, they never stop to spread in business environment. Running on mainframe at the beginning until they evaporate into the *Cloud*, Information Systems spread in all functional units of organizations: first in operation departments, then in accounting, marketing, human resources and so on. Nowadays, their presence goes far beyond the business world. Health care, education, government, all domains currently rely heavily on them. With the emergence of micro-computers, the Internet and now smartphones, they have also invaded individuals daily life. We use Information Systems for running, entertaining or even cooking (storing preferred recipes, computing calories, ordering ingredients...). IS are now ubiquitous.

Obviously, the information systems we are talking about are automatic systems which copiously rely on software and software engineering. It has become common to say that such software should be of high quality. By high quality, it is not only understood that it should be defect free, but first of all, it should ensure the satisfaction of the needs it was intended for. Indeed, when considering the ISO definition of quality, i.e. the "*degree to which a set of inherent characteristics fulfills requirements.*" [3], the focus is not put on the absence of *bugs* but the ability to satisfy requirements.

However, it was early identified that getting the requirements right is difficult [21]. Wrong selection of requirements has always been something to avoid since it is significantly more expensive to correct problems later in the development process [17]. It was one of the main purposes of waterfall methods as well as agile methods (although being diametrically opposed)[14]. This is also why so much effort were dedicated by academics and industry to what is called the Requirements Engineering (RE). This process is aimed at eliciting, modeling, and analyzing goals of a system in order to produce its specification. Pamela Zave described it as "*the branch of software engineering concerned with the real-world goals for, functions of and constraints on software systems. It is also concerned with the*

relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families" [156].

RE focuses on what is called the Requirements Problem (RP). Zave & Jackson provided an elegant formalization of this problem in their seminal paper [156] (hereafter called Zave & Jackson framework (ZJ)). They said that the RP amounts to find the specification S that, for given domain assumptions K, satisfies the given requirements R. Formally:

Definition 1.1 (Requirement Problem)**.**

$$K, S \vdash R \tag{1.1}$$

Jureta et al. pinpointed some limitations to this formulation of the RP [89]. First, it does not allows partial fulfillment of requirements. Secondly, there is no room for one specification to be better than another one. Consequently, we think that this formulation limits possibility to design high-quality systems.

For example, consider parents, whose children are traveling abroad, who want to talk with and see them. The ZJ framework focuses on finding a solution (a specification) considering the parent's requirements and some domain assumptions. A solution could be to install a video conference application on both children's smartphones and parents' laptop such that, given the assumption that they have wifi in the hotel, they could speak with them.

However, if there are no means to ensure video conference (because there is no wifi, they have no smartphone...), an acceptable solution could simply be a mobile call. Even if the requirements are not all satisfied, simple GSM conversation should still be an acceptable solution. ZJ framework lets no room for such reasoning and consequently partial solution.

The second criticism raises the fact that the ZJ framework is just focused on finding only one solution. However, going into an Internet café and start a Skype conversation is also a possible solution. This illustrates that instead of only one solution, there can exist several solutions for one particular problem. All possible solutions are what we call the *solution space*. When considering a solution space, we should be able to compare those solutions. Something that the ZJ framework does not integrate since it limits the RP to just find one point in this space and it says no word about criteria that would be used to compare solutions.

Several researchers highlighted that the concept of preferences should be introduced in the RP framework in order to compare several solutions [89, 100]. However, preferences allow only to get *ordinal* solutions. You can say that one solution is better than the other but there is no mean with ordinal solutions to say *how much* is a solution better than another. Moreover, it is often difficult to compare solutions on the basis of preferences on requirements: A can be preferred

to C, B can also be preferred to C but you could be in a situation where it is not possible to decide between A or B. Furthermore, too much preferences threaten their consistency (e.g. it becomes difficult to ensure transitivity between all possible solutions). Therefore, we think that it is relevant to formulate a RP framework which defines *cardinal* spaces based on an utility function, i.e. a space where you can say that one particular point is twice better than another.

We would like to stress another important limitation of the ZJ framework. The requirement problem (and consequently the RE process) is reduced to *find the specification*. Requirements are considered as given. This formulation removes the dimension that RE also determines the adequate set of requirements to consider. Still, RE is not just finding the specification but also *determining* the requirements. For the parent/children problem, engineers should be able to consider other requirements. Indeed, after discussing with the parents, they could abstract previous requirements into "be sure children are safe". Then, all new requirements could be identified such as "track children localization". It implies that the RP framework should also define and evaluate the *problem space*, what the ZJ framework explicitly rejects [156].

With the above criticisms, we do not mean that no progress have been made in RE research because all work would have been confined in this framework. For example, Goal Modeling (GM) has been deeply investigated as an interesting method to study the intention of the system, opening some doors regarding solution comparison (e.g. with Non-functional Requirements (NFR) [30]) or problem space definition (by goal refinement/absraction mechanism). A goal model states what should be satisfied by the system-to-be and refines it under sub goals until it specifies some tasks to do. This refinement process allows to consider several alternatives in the solution but also in the problem definition. As example, several approaches have been developed to study the variability resulting from this mechanism [64, 99].

In summary, we are interested in studying in this thesis a Requirement Problem formulation in which we define both *problem* and *solution* spaces in order to define a system which optimizes a particular cardinal *utility* function.

1.2 Research Scope: Requirement Optimization

From our previous discussion, we can introduce a new class of problem called the Requirements Optimization Problem (ROP) which consists in defining a tuple composed of domain assumptions, specifications and requirements optimizing stakeholder-defined criteria, formally:

Definition 1.2 (Requirement Optimization Problem)**.**

$$\text{maximize} \quad f(K,S,R) \tag{1.2}$$
$$\text{s.t.} \quad K,S \models R \tag{1.3}$$

The first modification this formulation brings is the use of an utility function able to compare different systems in order to find the optimal one (regarding the utility function). We also changed the $\vdash$ relation from ZJ framework, which stresses the focus on the *provability*, against $\models$ which models that we just want that K and S satisfy R. The focus on our RP formulation being more on the optimization than the provability. It highlights that rather than an attempt to replace the ZJ framework, our approach should be considered as a complementary framework.

From a mathematical point of view, our formulation directly refers to optimization problems which are defined as [20]:

Definition 1.3 (Linear Optimization Problem)**.**

$$\text{minimize} \quad f_0(x) \tag{1.4}$$
$$\text{s.t.} \quad f_i(x) \leq b_i, i = 1,\ldots,m \tag{1.5}$$

where

- the vector $x = (x_1,\ldots,x_n)$ is the optimization variables of the problem,
- the function $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the objective function,
- the functions $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1,\ldots,m$ are the (inequality) constraint functions,
- and the constants $b_1,\ldots,b_m$ are the limits, or bounds, for the constraints.

An optimal solution of this problem is a vector x^* if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $f_1(z) \leq b_1,\ldots,f_m(z) \leq b_m$, we have $f_0(z) \geq f_0(x^*)$.

Using optimization for software engineering is not new. It is called *Search-based Software Engineering* [69, 157] and has been used in several disciplines, e.g. project planning, cost estimation, testing or automated maintenance. Optimization of requirements has also been studied [112, 84]. Most of the time, it takes the form of a list of requirements described by value and cost for which we have to find the optimal subset.

However, those requirement optimization approaches are not rooted in any particular ontology for requirements engineering. We yet think that it is important to position the Requirement Optimization Problem in a particular ontology if

we want to benefit from all research progress that was previously made regarding this ontology. Indeed with a framework grounded in a requirements ontology, it should open our ROP framework to all concepts that were developed on the basis of this ontology such as its Requirements Modeling Language (RML).

In their work to define such an ontology (referred as Core Ontology for Requirements Engineering (CORE) hereafter) [89], Jureta et al. identified ontological requirement concepts as being goal, softgoal, quality constraint, domain assumption and task. We root our thesis in this ontology what brings us to the following definition of a *Requirement Optimization Problem in CORE*:

Definition 1.4 (Requirement Optimization Problem in CORE)**.**

$$\text{maximize} \quad f(K, T, Q, G) \tag{1.6}$$

$$\text{s.t.} \quad K, T \vDash Q, G \tag{1.7}$$

where

- G is the set of binary decision variables on goals satisfaction,
- S is the set of decision variables on softgoals,
- Q is the set of constants, constraints and decision variables (integer and real) used to model quality constraints satisfaction,
- T is the set of binary decision variables on tasks realization,
- K is the set of domain assumptions which covers domain statements, contribution links and any other concepts related to the domain. It can takes the form of constants, decision variables (integer, binary and real) and constraints.

However by doing this, we need to adopt another mathematical formulation of optimization because CORE requires also binary and integer variables while the model suggested in Definition 1.3 is only based on real variables.

Use of binary variables for goal satisfaction can be easily understood: the goal is satisfied or not. How it can be satisfied belongs to the scope of domain assumptions. If stakeholders specify that some task realization can contribute to the satisfaction of a goal, it is a domain assumption. Consequently, G does not have to integrate more than binary decision variables.

Regarding quality constraints Q, let's take as example the following statement: "Duration time for uploading a file should be less than 5 seconds." This statement is by itself an inequality *constraint* consisting of a *constant* (5 seconds) and a *real-decision variable* (the duration time).

From this discussion, we see that it is then necessary to integrate non-continuous variables (especially binary variables) into the mathematical model. The ROP would then rely on Mixed-Integer Programs (MIPs) which are optimization problems in which a nonempty subset of integer variables (binary variables being a specific case of integer variables) and a subset of real-valued variables exist, the constraints are all linear equations or inequalities, and the objective is a linear function to be minimized (or maximized) [153]. In mathematics, such problems are defined as:

Definition 1.5 (Mixed-Integer Optimization Problem)**.**

$$\begin{aligned} \text{minimize} \quad & f_a(x) + f_b(y) + f_c(z) && (1.8)\\ \text{s.t.} \quad & f_i^1(x) + f_i^2(y) + f_i^3(z) \leq b_i, i = 1,\ldots,q && (1.9)\\ & x \in \mathbb{R}^n \\ & y \in \mathbb{B}^m \\ & z \in \mathbb{Z}^o \end{aligned}$$

where

- the vector $x = (x_1,\ldots,x_n)$ is the real optimization variables,
- the vector $y = (y_1,\ldots,y_m)$ is the binary optimization variables,
- the vector $z = (z_1,\ldots,z_o)$ is the integer optimization variables,
- the functions $f_a : \mathbb{R}^n \to \mathbb{R}$, $f_b : \mathbb{B}^m \to \mathbb{R}$ and $f_c : \mathbb{Z}^o \to \mathbb{R}$ are the objective functions,
- the functions $f_i^1 : \mathbb{R}^n \to \mathbb{R}$, $f_i^2 : \mathbb{B}^m \to \mathbb{R}$ and $f_i^3 : \mathbb{Z}^o \to \mathbb{R}$ with $i = 1,\ldots,q$ are the (inequality) constraint functions,
- and the constants $b_1,\ldots,b_q$ are the limits, or bounds, for the constraints.

1.3 Research Problem: Formulation of ROP instances in CORE

With the given ROP formulation we provided in the previous section, different research questions arise. They are depicted in Tab. 1.1 and are individually described in this section.

First of all, it seems important to determine to which kind of problems Requirement Optimization can apply. Indeed, during RE several issues can appear:

Table 1.1: Research Questions

RQ1	Is there any RE issue with a non-trivial solution for which it is useful to formulate it as an optimization problem using concepts in CORE ?
RQ2	Given a RE issue formulated as a Requirement Optimization Problem, how can instances of this ROP be formulated with traditional RP models and is there any extension required to those models ?
RQ3	Given a specific ROP modeled with existing models, how can it be formulated as a Mixed-Integer Program ? In other words, what are the different decision variables, constraints and parameters to consider and how to transform the requirements model into a mixed-integer program ?

how to gather the information? how to clarify it? how to model it? Examples[1] of issues arising in RP solving as well as representative references are depicted in Tab. 1.2.

Given those RE issues, the first research question is to determine if there is any of the RE issues for which RO is relevant. This question is not about if optimization is relevant to RE issues; it is relevant and it has already been discussed by researchers (as we said in previous section). The question is rather if the formulation of ROP we provided in Definition 1.4 can be applied on any of those RE issues. This limits the scope of this research question to prove that some of the mentioned issues can be tackled as ROP rooted in CORE and not to exhaustively identify all issues in which ROP in CORE is relevant.

Research Question 1. *Is there any RE issue with a non-trivial solution for which it is useful to formulate it as an optimization problem using concepts in CORE?*

If the previous question is answered by the affirmative, then it becomes interesting to show how can ROP instances of this particular issue be modeled. The idea here being to rely on existing RE tools, in particular existing Requirements Modeling Languages. This implies that we should see if such existing modeling techniques can be used to model instances of ROP and if extensions are required.

[1] Those examples were identified during our readings of the RE literature. It was also significantly enriched by some research colleagues suggestions based on their own experience and readings. This table is not aimed to be exhaustive regarding the RE issues neither regarding the suggested references. It just proposes some significant RE research papers and illustrates the variety of the tackled issues in RE.

Table 1.2: Examples of RE issues arising in RP solving. Bold issues refers to problem treated in more details in this thesis as ROP.

#	Issue	Representative references
RI1	How to gather information about requirements, a system-to-be and its environment?	[62, 76, 41]
RI2	Which of the gathered information is relevant to RE?	[40, 155, 88]
RI3	How to clarify the elicited information, to avoid such problems as vagueness and ambiguity?	[108, 98, 87]
RI4	**How to determine which requirements have highest priority, for whom, and why?**	[91, 10, 75]
RI5	How to help stakeholders agree on common priorities over requirements?	[97, 16, 86]
RI6	**How to distribute the responsibility for the satisfaction of requirements to the system-to-be, systems it might interact with, and people in its environment?**	[40, 25, 53]
RI7	**How to estimate costs, risks, and deadlines for making systems that satisfy requirements?**	[13, 15, 135]
RI8	How to evaluate how complete requirements are, and if any important requirements may have been missed?	[72, 136, 160]
RI9	How to evaluate if the requirements are consistent, and to manage inconsistent requirements?	[146, 73, 52]
RI10	**How to specify and compare alternative strategies to satisfy the same requirements?**	[108, 98, 100]
RI11	How to evaluate the quality to which requirements would be satisfied by the system-to-be?	[108, 18, 95]
RI12	How to check if a system-to-be specification satisfies requirements?	[53, 61]
RI13	How to keep track of changes to requirements, reasons for these changes, their impact on existing requirements and systems, and how to propagate these changes in an existing specification of a system-to-be?	[66, 121, 33]
RI14	**How to do RE for systems that should adapt to different environments and requirements?**	[117, 35, 101]
RI15	How to do RE for systems that should autonomously adapt to their environment?	[29, 152, 22]

This step is necessary because modeling RP instances is difficult by itself and trying to write the mathematical model directly seems unpractical. It is then more convenient to allow requirement engineers to use available tools to model RP instances, allow them to integrate specific considerations for the optimization and then automatically generate the optimization mathematical model. This brings us to the second research question:

Research Question 2. *Given a RE issue formulated as a Requirement Optimization Problem, how can instances of this ROP be formulated with existing RML and is there any extension required to those RML ?*

Finally, once (and if) we have showed that a relevant ROP could be described with existing RML (which implies to define the objective function and what are K, T, S, Q, G in this models), it arises the question of how to formulate the ROP as a Mixed-Integer Program. In other words, what are the specific decision variables (related to K, T, S, Q and G) that should be considered and how to transform constraints or relations from the model in mathematical inequalities. This is the concern of our third research question:

Research Question 3. *Given a specific ROP modeled with existing models, how can it be formulated as a Mixed-Integer Program ? In other words, what are the different decision variables, constraints and parameters to consider and how to transform the requirements model into a mixed-integer program ?*

1.4 Research Contribution: Models, MIPs and POCs

This thesis brings four types of contributions into the field of ROP. Three types directly related to each research question previously described and one additional contribution related to the definition of a tool supporting contributions of **RQ2** and **RQ3**.

First, on the basis of a case study, we identified several situations where ROP are relevant: (i) determining a planning which maximizes customer expected value, (ii) selecting a solution which minimizes the cost, (iii) determining the optimal product portfolio in a software product line and (iv) determining the amount of configurability a solution should present. For each of those situations, we discuss why those issues should be considered as ROP and which criteria should be optimized. Those identified problems are all related to specific RE issues presented in Table 1.2.

The second type of contribution consists in formalizations of the identified ROP into requirements models. We show how instances of those problems can be modeled with existing RML. When required, we suggest some extensions to the

RML which take various forms: introduction of new concepts (nodes or relationships), introduction of new properties for existing concepts (e.g. a cost related to a task), mapping with other languages (e.g. using goal-oriented languages with features diagrams or entity-relationship diagrams).

Thirdly, for each identified ROP, we defined the underlying MIP and provided the mapping with the ROP model. Those mappings define how to translate requirement concepts and relations into decision variables and mathematical constraints.

Finally, a tool able to support the previously discussed contributions is provided. It allows to model RP instances with existing RML. It supports user-defined extensions of existing RML (for instance by adding new concept nodes or new labels on existing nodes) and also allows translation of RP instances into a MIP that can be executed by the tool.

1.5 Research Methodology: Design Science Research for IS

This research follows a methodology dedicated to design science research for Information Systems. Indeed, "*whereas natural sciences and social sciences try to understand reality, design science attempts to create things that serve human purposes*" [134]. That is what this thesis attempts to do. More precisely, we followed a methodology suggested by Peffers et al. [115] which consists of 6 steps: (1) Problem identification and motivating, (2) definition of the objectives for a solution, (3) design and development, (4) demonstration, (5) evaluation and (6) communication. This particular methodology has the advantage to synthesize previous Design Science Research Methodologies in IS such as work of Eekels et al. [46] or Hevner et al. [147].

The application of this methodology to our research is discussed in this section and is summarized in Fig. 1.1. But before describing each step, it is important to notice that our research is an *Objective-centered research*. It means that the entry point in the process was the definition of the objectives of the solution we wanted to provide: a Requirement Engineering framework which provides an optimal solution in the problem and solution spaces it defined.

Problem Identification and Motivating

Motivate our research of providing a framework for resolving ROP was done in two steps. First, we investigated the research literature in order to evaluate what were the different issues RE has to tackle and what was the state-of-the-art of RE regarding optimization of these issues. We did not exhaustively investigate all RE

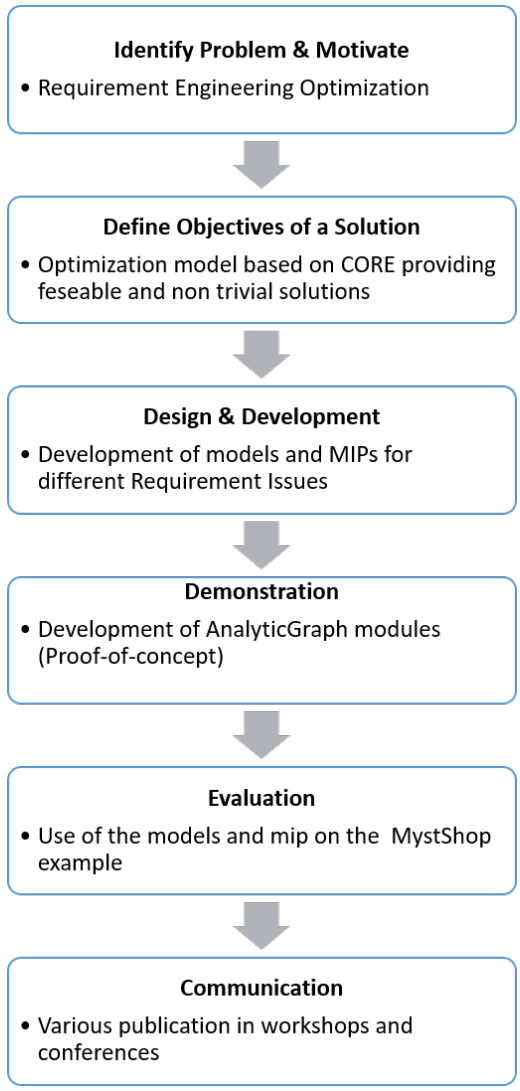


Figure 1.1: Description of the methodology followed in this thesis

issues but focused on **RI4**, **RI6**, **RI7**, **R10** and **R14**. This selection resulted from observed problems in our case study which will be presented in Chapter 2.

Definition of the Objectives for a Solution

The objectives of the intended solution have been discussed in previous sections and related research questions are depicted in Tab. 1.1. As a reminder, we are interested in studying in this thesis a Requirement Problem formulation in which we define both *problem* and *solution* spaces in order to define a system which optimizes a particular cardinal *utility* function.

Design and Development

The results of the design effort were extensions of some RML and the formulation of MIPs. This design required to make decisions on what criteria to take into account in order to formalize the objective function, in other words what should be optimized. The reasoning behind each of these decisions is presented in Part II, where each chapter is dedicated to specific RI.

Demonstration

Implementing our optimization model in a web-based tool called AnalyticGraph.com (AnalyticGraph) and applying it (a posteriori) on our case study demonstrated the feasibility and efficacy of our approach. This was done for each tackled Requirement Issue. During this step, some limitations were nonetheless identified such as MIP resolution duration that could become excessive depending on particular parameters. They are more discussed in Chapter 4.

Evaluation

Evaluate our approach would have required applying it (a priori) on another case study which was not done. It could be the purpose of further research.

Communication

Finally, results of this research were communicated in several conferences and workshops. All scientific papers are depicted in Tab. 1.3. Further publications in journals should be done.

1.6 Research Positioning: Deviation from the ZJ framework

In comparison with the ZJ framework, our work has a different positioning. Indeed, due to its specific formulation of the RP, the ZJ framework has epistemological consequences. Epistemology refers to the theory of knowledge, and in particular, how we acquire knowledge. Using a monotonic relation $\vdash$ in order to formulate the RP implies that all information is considered as existing *a priori* and should be discovered. We can (and therefore should) collect and analyze all requirements in the early phase of the project before going further. There is no room for additional information that could latter appear and would contradict previous inferences. It totally fits with the waterfall methodology. However, it omits the fact that all information acquired during RE comes from human speeches and/or constructs. It implies that some information could be omitted (deliberately or not), could become considered as false or true latter in the process. Moreover, requirements are never fully objective but firmly depends on stakeholders intentions, judgments, and values.

Moreover using the term *find a specification* suggests that, aside from the *a priori* existence of the solution, the role of the engineer is to discover it. It is rooted in a technical perspective of how RE should be performed. Engineer's role is mainly to prove that his specification will satisfy the requirements considered as given. We rather think that the solution is constructed (and not discovered) by engineers and stakeholders. By extension, if the solution is constructed, also is the problem. A formulation of the requirements problem should also take that into account. Engineer's role get a social dimension where he negotiates and constructs both the solution as well as the problem. This is why we prefer to use *define* instead of *find* in the RP formulation and we extent the formulation on both solution and problem spaces. Although it seems anecdotal, this is important for the research, education and practice in RE. One of the main consequences is that the ROP should rely on iterative and incremental processes and allows previous conclusions to be defeated. It also has an important influence on the role of the Requirement Engineer.

1.7 Thesis Outline

The present thesis divides into four parts:

Part I introduces the case study that is used through the thesis. It also describes the basic concepts on which this thesis is build. Among other it introduces Techne as the RML mainly used. Techne being defined using CORE, it makes it the most fitted RML for our purpose. This part also describes how to transform a Techne

model into a MIP. Eventually, before going into further details regarding ROPs, we discuss underlying assumptions and the derived limitations.

Part II presents four distinct RE issues for which we formalize a Requirement Optimization Problem, we describe how to model instances with Techne (and other existing languages if required) and we map the extended Techne model to a Mixed-Integer Program. Chapter 5 discusses the issue of providing an optimal planning. Chapter 6 presents the problem of finding the optimal balance between commonality and variability when developing information systems for different customers. Chapter 7 is about cost minimization. It suggests a ROP formulation using the Functional Point Counting to measure the effort required for each solution. Eventually, Chapter 8 presents self-configurable systems and suggest a trade-off model to determine the amount of configurability those systems should have.

Part III presents AnalyticGraph, the supporting tool we developed to support our contributions. It is an online platform in which all examples, MIP or RML extensions discussed in this thesis are accessible for free. Chapter 9 is dedicated to present how AnalyticGraph can support creation of RE models. It also shows how it improves previous RE existing software.

Part IV concludes this work by exploring , in Chapter 10, potential research tracks that, we think, could extent or improve the present work. Then, we provide a summary of the work and its main contributions.

Several chapters of this thesis resulted in publications. These latter are depicted in Tab. 1.3. Chapter 7 and the second part of Chapter 8 still need to be published.

Table 1.3: Summary of the publications and related chapters.

Paper reference	Related Chapter
J. Gillain, I. Jureta, and F. Stéphane. Planning optimal agile releases via requirements optimization. In *Third International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'16)*. Springer, 2016	Chapter 5
J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck. Product portfolio scope optimization based on features and goals. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 161–170. ACM, 2012	Chapter 6
C. Burnay, J. Gillain, I. J. Jureta, and S. Faulkner. On the definition of self-service systems. In *Advances in Conceptual Modeling*, pages 107–116. Springer, 2014	Chapter 8
J. Gillain, C. Burnay, I. Jureta, and S. Faulkner. Analytic-graph.com: Toward next generation requirements modeling and reasoning tools. In *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. IEEE, 2016	Chapter 9
J. Gillain, S. Faulkner, I. J. Jureta, and M. Snoeck. Using goals and customizable services to improve adaptability of process-based service compositions. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–9. IEEE, 2013	Briefly discussed in Chapter 10

2 Case Study

2.1 Introduction

In this chapter, we present the case study that will be used through this thesis in order to illustrate our contributions. This case covers the start of a Belgian mystery shopping company and focuses on the requirement issues it faced from its start in 2010 until 2015. The case contributes mainly to the first research question of this thesis, that is, the identification of requirement issues that could relevantly be formulated as an optimization problem. Indeed, as we will present, the company faced different RE issues at different stages of its development. Those specific RE issues often presented trade-off situations which were used as starting points for ROP formalization.

Solutions suggested in this thesis were not validated with application on this case since they were designed *a posteriori* of the problem occurrence. Nonetheless, this case contributed to the following work by answering these questions:

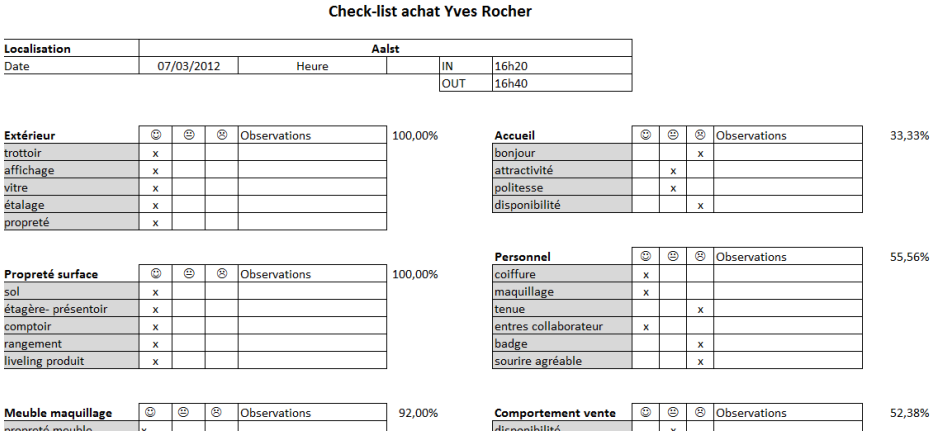
- What are the trade-off problems of requirement engineering an organization could face ?
- Can those problems be formalized as optimization problems ?
- Does it bring a non-trivial solution?

In the next section, we present a brief history of the case study as well as information systems that were implemented. We summarize in section 2.3 the different optimization problems identified in the case study.

2.2 History

ASC Concept is a small Walloon mystery shopping company founded in 2010. At the beginning, the organization mission was to provide three types of services: mystery shopping, customer service training and customer satisfaction surveys.

The society started with 3 customers for mystery shopping services:



Check-list achat Yves Rocher

Localisation	Aalst			
Date	07/03/2012	Heure	IN	16h20
			OUT	16h40

Extérieur	☺	😐	☹	Observations	100,00%
trottoir	x				
affichage	x				
vitre	x				
étalage	x				
propreté	x				

Accueil	☺	😐	☹	Observations	33,33%
bonjour			x		
attractivité		x			
politesse		x			
disponibilité			x		

Propreté surface	☺	😐	☹	Observations	100,00%
sol	x				
étagère- présentoir	x				
comptoir	x				
rangement	x				
liveling produit	x				

Personnel	☺	😐	☹	Observations	55,56%
coiffure	x				
maquillage	x				
tenue			x		
entres collaborateur	x				
badge			x		
sourire agréable			x		

Meuble maquillage	☺	😐	☹	Observations	92,00%

Comportement vente	☺	😐	☹	Observations	52,38%

Figure 2.1: Example of the Excel checklist used to evaluate stores during mystery visits.

- **Night&Day:** late night grocery shops,
- **Yves-Rocher:** beauty supply stores,
- **Mestdagh:** supermarkets,

Each customers had between 30 to 50 stores to be checked at a frequency varying between 6 to 12 times a year. The provided service consisted in visiting periodically the different stores and evaluate each of them following a checklist of predefined criteria. An additional written report of one page was also provided. Checklists were filled in with an Excel spreadsheet and then send by email with the written report. The different criteria to be checked were grouped by shelves and a three level-scale was used to evaluate each criteria. Additional comments could be added to each checked criteria. An example of this criteria list is provided in Fig. 2.1.

In addition to the visit checklists, ASC Concept also provided its customer with a table summarizing by store the average score of each shelve group for all previously performed visits. The summarizing table were maintained manually by the company employees. An example of this table is provided in Fig. 2.2. Finally, at the end of the year, the supermarket company also asked for an additional table summarizing the global score for all stores. This table being used to reward the best store.

At the end of 2010, ASC Concept decided to create a website in order to improve its visibility. The website purpose was limited to the activity presentation. In order to realize the website, ASC Concept contacted a small IT company.

Belgrade

	Global	Parking-sas entrée	Fruits et légumes	Char.ls/crèm /TDLM/vola	Poisson	Fromage coupe	Charcuterie coupe	Bouch./Ls/ gastronome	PGC-surgelés-zone event.	Boulangerie	Caisse	Easy shopping-image				
3ème salve 2012	73,47%	64,44%	80,95%	55,56%	73,33%	62,96%	100%	74,60%	100%	100%	40,74%	55,56%				
4ème salve 2012	71,60%	82,22%	68,25%	64,44%	64,44%	55,56%	73,33%	68,25%	88,89%	82,22%	66,67%	73,33%				
5ème salve 2012	83,20%	91,11%	74,60%	73,33%	100%	77,78%	82,22%	87,30%	77,78%	100%	77,78%	73,33%				
6ème salve 2012	75,98%	91,11%	74,60%	73,33%	91,11%	62,96%	82,22%	61,90%	77,78%	91,11%	66,67%	62,96%				
	Global	Parking-sas entrée	Fruits et légumes	Charcuterie Ls	Volaille	Crèmerie	Poisson	Fromage coupe	Charcuterie coupe	Boucherie + gastronome	Boucherie Ls	Surgelés	Boulangerie	Caisse	Easy-shopping-image globale	Pub relance
7ème salve 2012	82,95%	93,65%	85,71%	100,00%	75,00%	100%	80,00%	75,93%	80,00%	79,37%	100%	77,78%	80,00%	80,00%	80,00%	
8ème salve 2012	70,80%	79,37%	66,67%	66,67%	77,78%	100%	51,11%	68,52%	80,00%	60,32%	77,78%	63,89%	62,22%	80,00%	100%	
1e salve 2013	84,42%	93,65%	85,71%	100%	52,78%	100%	80,00%	83,33%	80,00%	73,02%	100%	100%	91,11%	71,11%	71,11%	91,11%
2e salve 2013	71,73%	100%	73,02%	37,04%	75,00%	100%	91,11%	53,70%	80,00%	46,03%	75,00%	100%	51,11%	60,00%	62,22%	64,44%
3e salve 2013	79,12%	93,65%	85,71%	66,67%	63,89%	85,19%	100%	61,11%	80,00%	60,32%	77,78%	100%	100%	51,11%	82,22%	62,22%

Figure 2.2: Example of the Excel spreadsheet summarizing global scores of each store.

Once the website was developed, ASC Concept asked the IT company to develop an access-limited webplatform where it would be possible to upload the different reports. Specific access to the platform was provided to each customer. First requirements were quite simple. An employee of ASC Concept should be able to upload a report by specifying the customer, the store and the report type (i.e. a visit checklist, a visit written report or a summary spreadsheet). Old reports could be archived. For each customer, stores could be added, modified or archived as well as new access accounts could be created and linked to this particular customer. A screenshot of the web application is provided in Fig. This version required approximately 120 man-hours of development. 2.3.

At this time, the IT Team was asked by a property management company to develop a platform where it was possible to upload documents related to the properties (e.g. invoices for shared local maintenance). Document access needed to be limited to granted persons. This situation rises the possibility to reuse the created platform for ASC Concept as a basis for this new customer. It finally never materializes.

Until 2012, ASC Concept requirements had remained quite stable when an important business fact implied some changes. Contracts with the cosmetic stores and grocery shops ended while the store numbers to be visited for the supermarket company dramatically increased. This increase largely compensated the loss. The supermarkets' needs regarding analytics increased while the written report was given up. In order to ease the analytics generation which consumed a lot of time, ASC Concept decided to automatize this process. Two solutions were considered at the time:

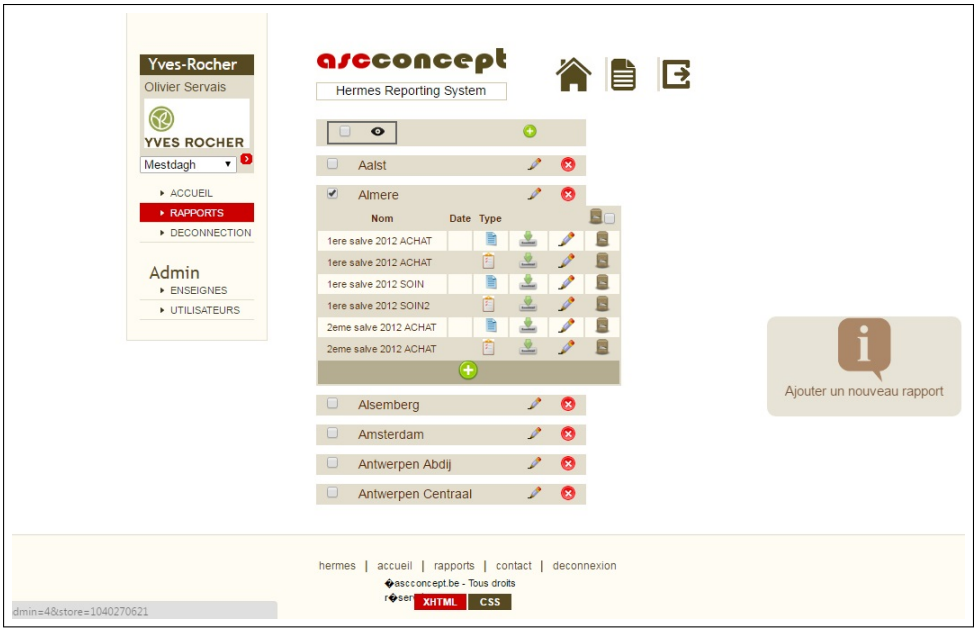


Figure 2.3: Screenshot of the first webapplication developed for ASC Concept.

1. Continue to use the current platform with the spreadsheet and create a VBA script able to extract data from the Excel files,

2. Create a webapp with the checklist to be filled online and then stored in a database allowing both visit result viewing as well as analytics computation. Being web-based, results and analytics would be accessible in a browser (and consequently on tablets or smartphones).

Although much more expensive for the company, the second solution was selected. It was justified mainly for three reasons. First, it would ease daily operations regarding the company activity. Secondly, it would improve company image thanks to more professional services. Thirdly, the cost could be reduced thanks to Walloon government subsidies dedicated to Internet-based applications.

Due to time and budget limitations, the platform was designed for a single customer with an unique checklist. At the early stage of this new information system, an employee of ASC Concept could fill results of a visit in a webform (depicted in Fig. 2.4). Contacts of the supermarket stores could access visit results as well as summary dashboards (depicted in Fig. 2.5). Moreover, due to its growth, ASC Concept was required to hire some external mystery shoppers. In order to be sure that visits results were correctly filled in, ASC Concept asked to distinguish

Figure 2.4: The checklist of the web-based application.

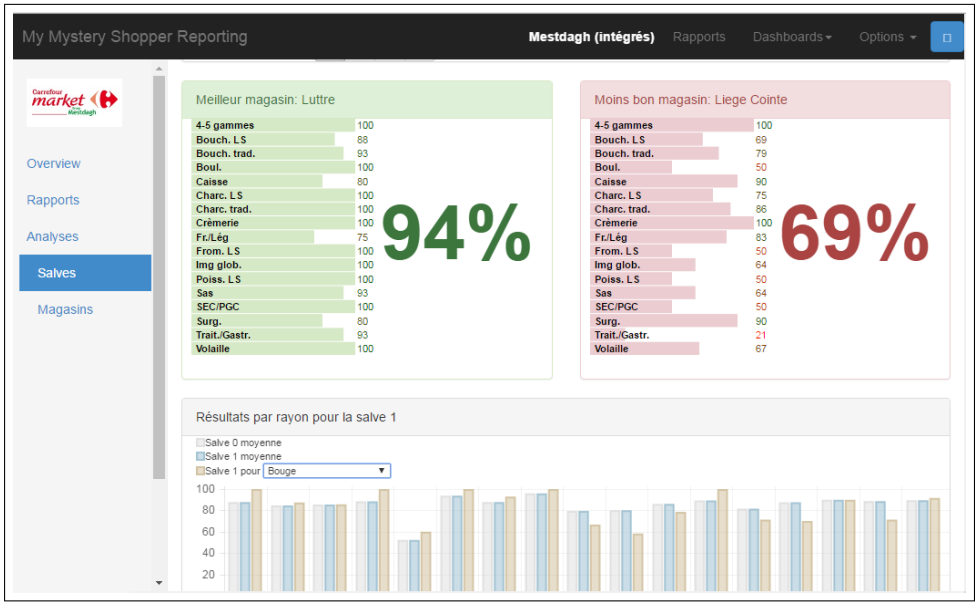


Figure 2.5: Analytics provided by the web-based application.

administrator accounts and mystery shopper accounts. The fill-in process became a two step process consisting of filling and then validating the report. This new version of the platform required 250 man-hours of work.

During one year of activity, few changes were recorded. The only requests from ASC Concept to the IT company was to add a contact account for the customer (e.g. a new sales manager) or to add/remove some criteria from the checklist. Since this occurred seldom, no administration interface was designed for it, insertion occurring directly in the database. A maintenance of approximately 1 man-day per month was necessary.

After this first year using the new application, ASC Concept gets a new contract with *franchised* stores from the same supermarket brand (until there, only the *integrated* stores were visited). Having separate management, results of the integrated stores should not be visible for the franchised stores and reciprocally. The franchised stores accepted to use the same criteria list. The only new feature was the possibility to manage different customers in the platform brands (but there was still only a single checklist common to all customers). This new feature required about 3 man-days to be integrated.

After an additional year of activity with the same question list for the two separate customers, some changes were required from both customers. It was decided to integrate the possibility to have a dedicated question list for each customer. Since there were only two question lists, no questionnaire designer was developed, the question list being inserted by developers in the database. It requires 4 additional man-days.

In 2014, the company decided to focus its activities on the mystery shopping and give up all other services (i.e. training and satisfaction surveys). It changes his name into MyMysteryShopper (hereafter called MystShop). The supermarket company asked some change in the application. First, it needed to provide to each store manager an access to his own store results. So, instead of having only global access for all store results, each store manager would be able to view its own results without accessing others results (brand manager being still able to access results of all stores of the brand). Moreover, it also asked to be warned by mail of any new visit report on the website. Eventually, it asked to have print-friendly template of visit reports.

During this period, two additional companies were prospected by MystShop: a kitchen selling company and an underwear brand company. In order to convince these potential customers, MystShop asked to implement custom checklist to be presented to the prospects. The IT Team being busy with the new store manager access and the warning system, this situation rises the question of a checklist designer. Indeed, being annoyed by the time required by the IT Team to design and implement new checklists, administrators of MystShop rise the possibility

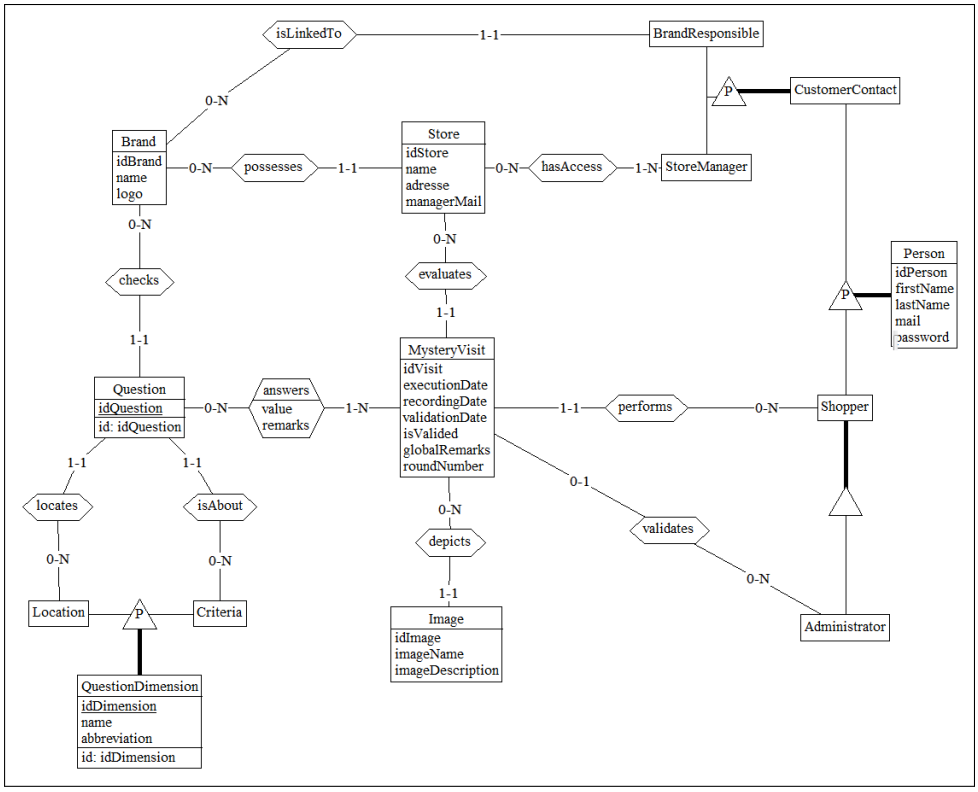


Figure 2.6: E-R diagram of current MystShop application data structure.

of designing the questionnaire them selves in order to prospect new customers. A checklist designer was eventually developed and accessible to administrators only.

In the current state, the application allows the following use case. The entity-relationship diagram of the application is depicted in Fig. 2.6.

2.3 Identified Requirement Issues

Regarding the previously described case, MystShop faced many requirement problems. Those problems are not new and have already been studied by the research community. For some of them, there already exist some attempts to solve them. The purpose of this thesis is to approach them from an optimization perspective and capitalize on what has been done before.

That being said, a first temptation when looking at those previously mentioned

requirement issues, could be to argue that applying a waterfall process starting with an in-depth requirements analysis could have avoid many problems. But it would have required that MystShop clearly knew *a priori* all its requirements. However, many unpredictable changes guided the evolution of MystShop's requirements (gain/loss of customers, focus on mystery shopping and withdrawal of training activities...). Moreover, MystShop's available resources at the beginning was quite limited forcing it to reduce the IS scope to the minimum which maximized return on investment. Eventually, many features were required in a short opportunity window which implied to focus on the very urgent requirements and leaving no room for comprehensive requirement analysis.

Nonetheless, when facing each of those RE issues, MystShop was confronted to multiple solutions. The choice was guided by informal reasoning processes on the basis of unclear criteria. In this thesis, we decided to approach them with a more formal optimization perspective which formulates both the selection criteria and the decision process. Here are a summary of the tackled problems with references to the Requirements Issues (RIs) discussed in Tab. 1.2:

- Due to limited resource, the IT team could not deliver all required features in the desired time. Three approaches were investigated to tackle this problem.
  - First, several solutions were considered in order to minimize its cost. This problem was formalized in Chapter 3 and a more comprehensive approach is suggested in Chapter 7. Finding the alternatives which minimize the cost relates to **RI10** and **RI7**.
  - Then, the specification of some priorities on the requirements in order to set a release planning following those priorities was considered. This problem is known as the Next Release Problem [7]. We present our contribution to this optimization problem in Chapter 5. This issue relates to **RI4** and also **RI7**.
  - The third considered approach was to transfer some design responsibility from the IT team to the business users. This is the case for the checklist designer. Our contribution (related to **RI6**) is presented in Chapter 8 and it deals with **RI6**.
- The IT company faced the possibility to reuse parts of the web application for another context. This reuse problem has been deeply studied by the Software Product Line Community. Our contribution to formalize this problem **RI14** as an optimization problem is described in Chapters 6 and 7.

3 Mapping Techne on a MIP

In this Chapter, we describe in details how to transform a goal-model described with the Requirements Modeling Language *Techne* into a Mixed-Integer Program. As it is, this chapter already contributes to answer research questions since:

- it suggests a model to compare alternative strategies to satisfy some requirements (**RI10**) and briefly discusses cost estimation (**RI7**) since alternative strategies are compared, among other factors, on the basis of their cost,
- it sets the foundation of the mapping between RP models and Mixed-Integer Programs that will be used through all the rest of this thesis (i.e. for each specific RE issue),
- Techne being a RML based on the CORE ontology, it respects our ROP formulation given in Def. 1.4.

This chapter divides into three parts. In section 3.1, we present Techne [85], the Goal Modeling Language that we will use through this thesis in order to model the ROP instances. Secondly, in section 3.2, we present all sets, parameters, decision variables and constraints required to map Techne to a MIP. Finally, in section 3.3, we introduce AnalyticGraph.com, the web-based application supporting the suggested approaches presented in this thesis.

3.1 Description of the Requirement Modeling Language Techne

Techne is described as *"an abstract requirements modeling language that lays formal foundations for [. . .] modeling languages applicable during early phases of the requirements engineering process."* [85]. Three reasons motivates our choice of Techne as the selected RML. First, Techne is an abstract goal modeling language based on the CORE ontology. It makes it applicable in a large spectrum of requirement problems. Secondly, it is straightforward how to translate Techne models into models in other goal modeling languages [154, 144]. It implies that conclusions and models suggested in this report can be applied with other languages. Finally,

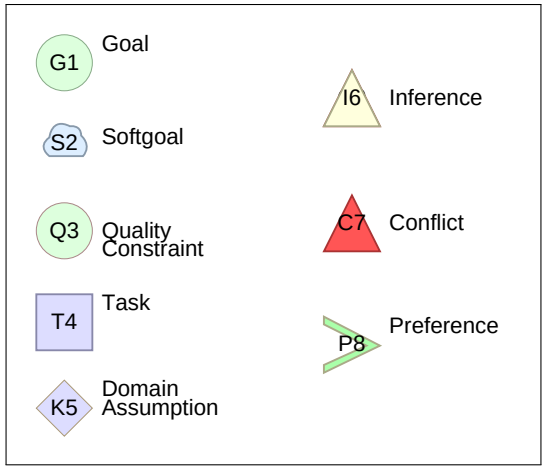


Figure 3.1: Graphical notation of Techne Nodes

because of its reasoning rules (e.g. *Inference node* is based on the *modus ponens* inference rule) Techne can be quite easily mapped to a MIP.

There are two types of primitive in Techne: concepts (goal, task, softgoal, quality constraint and domain assumption) and relations (inference, conflict, preference, is-optional and is-mandatory). These primitives are respectively aimed at classifying and specifying relations between the collected information during requirements elicitation. When used for modeling instance of a RP, both concepts and relations are depicted as *labeled nodes* which can be connected to each other with *directed links*. Those links have no label and consenquently no distinguishable meaning. It means that a conflict relation between two concepts will be depicted with three nodes, i.e. two nodes for the concepts and one for the relation.

Techne does not come with a graphical notation. For convenience, we adopt the notation depicted in Fig. 3.1. For modeling ease, we decided to not represent is-optional and is-mandatory relations as nodes but as additional labels on related nodes. In order to describe and illustrate those concepts, an example of a goal model formalized with Techne is provided in Fig. 3.2. It is based on the MystShop case study [1]. Here is the description of each Techne primitive [85]:

Goal:

Stakeholder desires become instances of the goal concept, if they refer to conditions, the satisfaction of which is desired, binary and verifiable. It is

[1] This example is accessible to `http://analyticgraph.com/dev/?g=Go1g0rLDx0`

represented by a circle labeled "G". As an example G0 is a goal which states that "Mystery visit results should be made available to the customer".

Quality constraint:
Stakeholder desires become instances of the quality constraint concept, if they refer to conditions, the satisfaction of which is desired, non-binary and verifiable. It is represented by a circle labeled "Q". An example of a quality constraint could be "Duration time for uploading a mystery visit report should be lesser than 5 seconds."

Softgoal:
Stakeholder desires become instances of the softgoal concept, if they refer to conditions, the satisfaction of which is desired, vaguely constraint and not necessary directly mesurable. It is represented by a cloud labeled "S". As an example S107 is a softgoal which states that "Mystery visit results are accessible everywhere".

Task:
Stakeholder intentions to act in specific ways become instances of tasks to be accomplished either by the system-to-be, or in cooperation with it, or by stakeholders themselves. It is represented by a square labeled "T". As an example T4 is a task which states the intention of developers to "Deploy a webplatform".

Domain assumption:
Beliefs are instances of domain assumption, stating conditions within which the system-to-be will be performing tasks in order to achieve the goals, quality constraints, and satisfy as best as feasible the softgoals. It is represented by a diamond labeled "K". As an example K12 is a domain assumption which states the "Regional government subsidises Internet-based services".

Inference:
This relation conveys the idea that a requirement can be the immediate consequence of another set of requirements. Since Techne considers that goal refinement and task decomposition ask basically the same problem, this relation is used to depict both relations. It is represented by a yellow triangle labeled "I". Incoming nodes are called *"premises"* while the outgoing node (which is unique) is called the *"conclusion"*. As an example I1 states that the conclusion G0 "Mystery visit results should be made available to the customer" will be satisfied if both premises G5 "Visit reports are recorded in a spreadsheet" and G6 "Visit reports are sent by email" are satisfied.

Conflict:
The conflict relation states that if conflict members (two or more) of this relation are satisfied in the solution, this latter will be inconsistent. Consequently, a solution should be conflict-free. It is represented by a red triangle labeled "C". It can have only incoming nodes (its members). As an example, nodes G6 and G16 are in conflict (depicted by node C94) which means that a solution where reports are sent by mail can not ensure that they are available in a central repository.

Preference:
Stakeholder evaluations of requirements convey that not all requirements are equally desirable. If a requirement is strictly more desirable than another one, then there is a preference relation between them and by strictly, we mean that they cannot be equally desirable. The preference relation leads to *ordinal* solutions, i.e. solutions that can only be ordered but for which it is impossible to compare how much it is preferable than another one. However, it is important to remind that we want *cardinal* solutions in our optimization framework. Then, the preference relation would not be necessary in our framework.

isMandatory/isOptional:
The is-mandatory relation on a requirement indicates that the requirement must be satisfied, or equivalently, that a conflict-free set of requirements which does not include that requirement cannot be a candidate solution. In contrast to the is-mandatory relation, the is-optional relation on a requirement indicates that it would be desirable for a conflict-free set of requirements to include that requirement, but that set can still be a candidate solution if it fails to include the optional requirement.

3.2 Mapping between Techne and MIP

In this section, we present a first mapping between Techne and a MIP. We first focus on concept nodes, then we describe how to map relation nodes.

Concept nodes

When mapping concept nodes to a mathematical model, there are two important aspects to consider. First, we need to be able to make references to specific nodes in the goal model. We do this by defining several **sets** which relate to some nodes in the goal model. Knowing that we have 5 different types of concept nodes

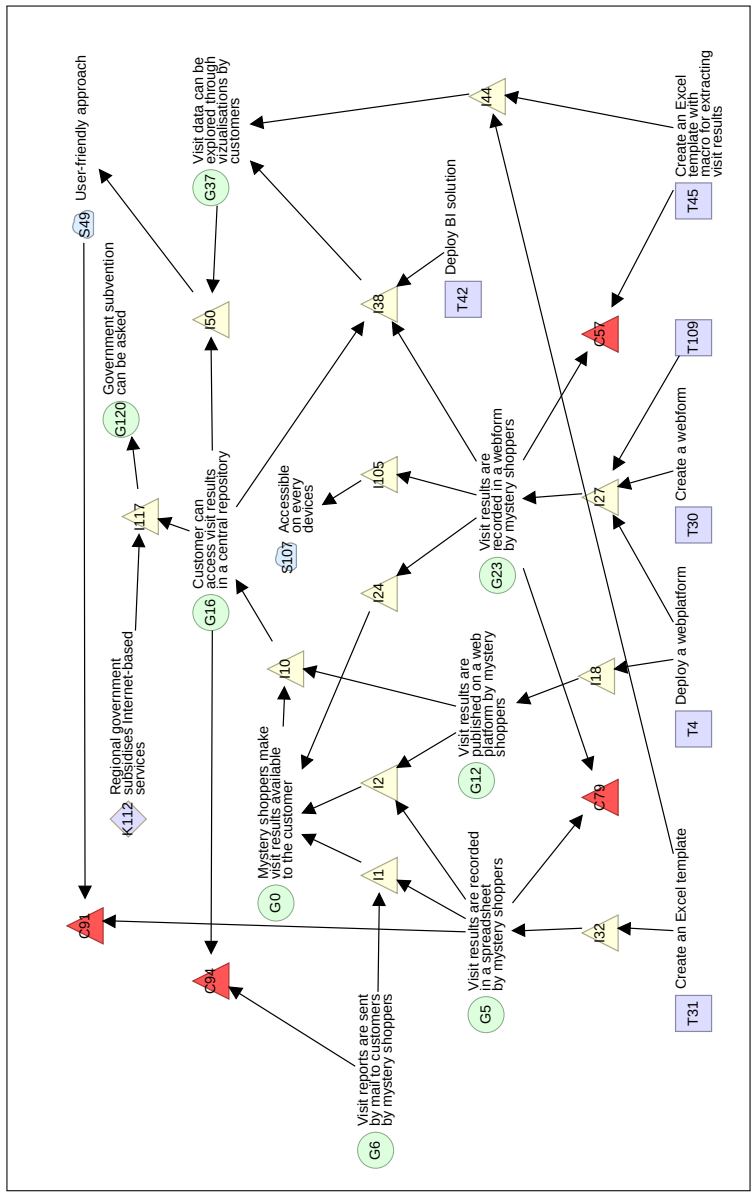


Figure 3.2: Modeling of the Mystery Shopper Case. Model accessible and editable at `http://www.analyticgraph.com/dev/?g=Go1g0rLDx0`

in Techne, namely goals, softgoals, quality constraints, domain assumption and tasks, we respectively define 5 sets: G, S, Q, K and T with

$$N = G \cup S \cup Q \cup K \cup T \tag{3.1}$$

Then, we define for each concept node (except for quality constraints) a binary **decision variable** in the mathematical model. When the decision variable value equals 1, it means that the concept is satisfied, when its value is set to 0 it means that it is not satisfied.

$$\forall n \in N \setminus Q : \sigma_n \in \mathbb{B} \tag{3.2}$$

For example, if σ_{G5} is set to 1, it states that "Visits results are recorded in a spreadsheet by mystery shoppers".

However, quality constraints are not mapped to binary decision variables but rather to constraints and integers or reals. Consider the following quality constraint Q1: *average time an ambulance spend to reach an accident location should be less than 200 sec*. Now, let us consider that we identified two tasks able to reduce this average time: T1 *Replace old Renault vehicles with high-speed BMW* and T2 *Install a fire pole*. If we consider that the current average time is 220 and that the tasks can respectively reduce the time by 40s and 30s, we have the following quality constraint:

$$220 - 40\sigma_{\mathsf{T1}} - 30\sigma_{\mathsf{T2}} = \sigma_{\mathsf{Q1}}$$

Then, the value of Q1 can have a particular utility function (which is used in other constraints and/or in the objective function). However, this approach has several limitations:

- if the duration time contributes to the objective function, the relation between the objective value and the decision variable σ_{Q1} is always linear,
- although we can show that it is possible to model non additive relations between the tasks and σ_{Q1}, it requires the introduction in the model of additional binary decision variables. For example, if the combined use of T1 and T2 reduces the average time of 60s instead of 70s, the mathematical constraints should be:

$$220 - 40\sigma_{\mathsf{T1}} - 30\sigma_{\mathsf{T2}} + 10\sigma_{\mathsf{x}} \leq \sigma_{\mathsf{Q1}} \tag{3.3}$$

$$\frac{\sigma_{\mathsf{T1}} + \sigma_{\mathsf{T2}}}{2} \geq \sigma_{\mathsf{x}} \tag{3.4}$$

where σ_{x} is a binary decision variable representing the simultaneous use of both σ_{T1} and σ_{T2}.

Then, for all quality constraint nodes in the model, we have at least both a mathematical constraint and a decision variable defined on real or integer:

$$\forall q \in Q : \sigma_q \in \mathbb{R} \oplus \sigma_q \in \mathbb{N} \tag{3.5}$$

$$\forall q \in Q : \mathsf{c} + \sum_{x \in \mathsf{in}(q)} \sigma_x * w_{x,q} = \sigma_q \tag{3.6}$$

where $w_{x,q}$ is the contribution of the satisfaction of the node x to the quality constraint q, e.g. it is -40 for T1 in our previous example. $\mathsf{in}(q)$ is the set of incoming nodes of q and c is a constant.

Additional consideration as node labels

In order to make the optimization, it is necessary to introduce additional considerations that will be part of the optimization criteria. Such criteria is introduced by defining a label on particular nodes (i.e. on all goals) and specifying for each individual goal a specific value. For example, we could introduce the notion of utility in the goal-model by defining an *utility function* noted $\mathsf{u}(x)$. For goals, this utility value has to be considered as expected revenue resulting from their satisfaction. For instance, if goal G16 *"Customer can access visit results in a central repository"* is satisfied, it should leverage 500 units of utility[2]. Reasoning is similar for softgoals and quality constraints. For tasks, utility should better be considered as a cost (and being negative). For instance, implementing task T4 "Deploy a webplatform", for example, will cost 200. Formally, u is defined on set N and has real as images.

$$\mathsf{u} : N \to \mathbb{R} \tag{3.7}$$

Objective function

In all MIP, we will define an objective function that the program will maximize or minimize. This objective function will be specific for each problem tackled by the model. As an illustration in this basic model, we define the objective function as being the maximization of the utility achievable by the model.

$$\max \sum_{x \in N} (\mathsf{u}(x) * \sigma_x) \tag{3.8}$$

[2]How to estimate the utility value of a goal is out of the scope of this thesis. Still, a brief discussion about this problem is provided in the next chapter.

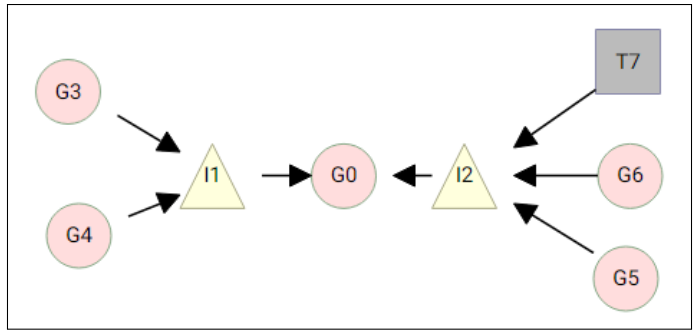


Figure 3.3: Example of goal model

Relation nodes

We will now discuss how we map semantics of relation nodes into a MIP. This takes the form of several sets of constraints and additional decision nodes. However, before going further into the definition of those constraints sets, it is useful to define two additional functions, namely *incoming nodes* and *typed incoming nodes*. Let us denote incoming nodes of node x as $\mathsf{in}(x)$ with

$$\mathsf{in}(x) \subseteq N \tag{3.9}$$

By extension, we defined as incoming node of a particular type, let say incoming inferences of node g, the following function:

$$\mathsf{in}_Y(x) = \mathsf{in}(x) \cap Y \tag{3.10}$$

In order to illustrate these functions, consider the example depicted in Fig. 3.3.

$$\mathsf{in}(\mathsf{G0}) = \{\mathsf{I1}, \mathsf{I2}\} \tag{3.11}$$

$$\mathsf{in}(\mathsf{I2}) = \{\mathsf{G5}, \mathsf{G6}, \mathsf{T7}\} \tag{3.12}$$

$$\mathsf{in}_G(\mathsf{I2}) = \{\mathsf{G5}, \mathsf{G6}\} \tag{3.13}$$

Mapping an inference node

An inference node is mapped into the MIP as a decision variable and two additional constraints in the mathematical model. Those constraints are called the *premises constraint* and the *conclusion constraint*. For instance, let us consider the

inference I2 in Fig. 3.3. This inference node I2 will be satisfied iff all premises nodes are also satisfied. It can be translated into decision variables as:

$$\sigma_{\mathsf{I2}} \leq \frac{\sigma_{\mathsf{G5}} + \sigma_{\mathsf{G6}} + \sigma_{\mathsf{T7}}}{3} \tag{3.14}$$

In other words, I2 can be considered as satisfied (i.e. $\sigma_{\mathsf{I2}} = 1$) only if premises nodes (i.e. G5, G6 and T7) are all of them also satisfied (i.e. $\sigma_{\mathsf{G5}} = \sigma_{\mathsf{G6}} = \sigma_{\mathsf{T7}} = 1$). This is the *premises constraint* of I2. For all inference nodes, it can be generalized as:

$$\forall i \in I : \sigma_i \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_x}{|\mathsf{in}(i)|} \tag{3.15}$$

where $|\mathsf{in}(i)|$ is the cardinality of set $\mathsf{in}(i)$.

The second set of constraints is defined as *conclusion constraints*. They say that all concept nodes which have incoming inferences can be satisfied only if at least one of the incoming inference is also satisfied.

$$\forall n \in \{x | x \in N \wedge \mathsf{in}_I(x) \geq 0\} : \sigma_n \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_i \tag{3.16}$$

Mapping a conflict node

Two concept nodes (e.g. goal nodes) can be in conflict. This means that those goals cannot be satisfied together. For example consider C94 which states that nodes G16 *Customer can access visit results in a central repository* and G6 *Visit reports are sent by mails to customers* are incompatible.

Let us consider the set of conflict nodes C. There are *conflict constraints*

$$\forall c \in C : \sum_{\sigma_x \in \mathsf{in}(c)} x \leq 1 \tag{3.17}$$

Optional vs. Mandatory Nodes

A concept node which is mandatory must be satisfied in the solution. In other words, the solution must imperatively have the value of its corresponding decision variable to 1.

It is translated in the following equality constraint:

$$\forall n \in M : \sigma_n = 1 \tag{3.18}$$

All other nodes which are not mandatory (i.e. optional nodes) do not require particular constraints.

MIP 3.1. Description of the basic MIP for a Techne goal model.

Sets

G	$= \{g_0, \ldots, g_i\}$	Goal nodes
M	$\subseteq N$	Mandatory nodes
T	$= \{t_0, \ldots, t_i\}$	Task nodes
S	$= \{s_0, \ldots, s_i\}$	Softgoal nodes
Q	$= \{q_0, \ldots, q_i\}$	Quality Constraint nodes
K	$= \{k_0, \ldots, k_i\}$	Domain assumption nodes
N	$= G \cup T \cup S \cup K$	Concept nodes
I	$= \{i_0, \ldots, i_i\}$	Inference nodes
C	$= \{c_0, \ldots, c_i\}$	Conflict nodes
N^*	$= N \cup I \cup C \setminus Q$	Binary-based satisfaction nodes

Decision Variable

σ_{N^*}	$= \{\sigma_i \in \mathbb{B} : i \in N^*\}$	Binary variables representing node satisfaction.

Functions

u	$: N \to \mathbb{R}$	Utility function
$\mathsf{in}(x)$	$\subseteq N^*$	Incoming nodes function
$\mathsf{in}_Y(x)$	$= \mathsf{in}(x) \cap Y$	Typed incoming nodes function

Objective function

$$\max \sum_{n \in N} \mathsf{u}(n) * \sigma_n$$

Constraints

Premises constraints
$\forall i \in I$:
$$\sigma_i \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_x}{|\mathsf{in}(i)|}$$

Conclusion constraints
$\forall n \in \{x \in N : |\mathsf{in}_I(x)| > 0\}$:
$$\sigma_n \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_i$$

Conflict constraints
$\forall c \in C$:
$$\sum_{x \in \mathsf{in}(c)} \sigma_{x,p} \leq 1$$

Mandatory constraints
$\forall i \in M$:
$$1 \geq \sum_{p \in P} \sigma_{i,p}$$

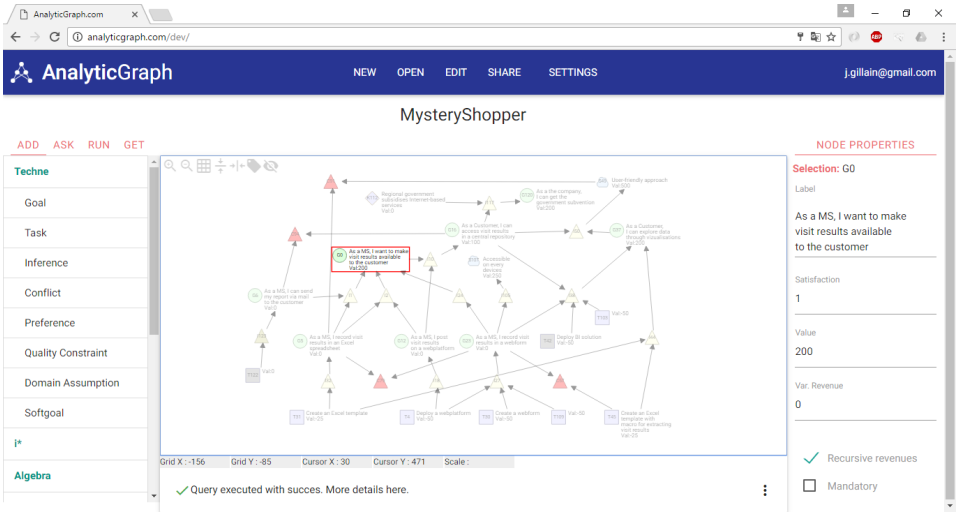


Figure 3.4: Workspace of AnalyticGraph.com

3.3 AnalyticGraph.com for Simple ROP

In order to support the whole approach presented in this thesis, a tool named AnalyticGraph was developed. The tool supports modeling of goal diagrams in Techne (or other RML) and runs the optimization underlying the requirement problem. Tutorials on how to use it are provided [3]. Examples used in this thesis are also available via urls. Part III of this thesis being dedicated to AnalyticGraph.com, more information can be found there.

Figure 3.4 shows a screenshot of the workbench of AnalyticGraph.com with the MystShop case. Techne primitives are accessible on the left panel, while the right panel allows to specify some labels attached to the selected node (e.g. utility, the mandatory aspect).

[3] http://analyticgraph.com/tutorial-on-the-optimization-of-goal-models/

4 Limitations

Before focusing on specific ROPs, we will discuss in more details assumptions used in our approach and consequently the resulting limitations. Those assumptions and limitations are general to our approach, the more specific ones (i.e. those related to each model) will be discussed within the dedicated chapters. Such discussion is nonetheless important to precisely scope the present work. We also discuss how the identified limitations can be mitigated or what research should be conducted in order to tackle them.

4.1 Validation and Value for Requirements Engineering

The first limitation of this thesis is its lack of validation. Indeed, despite we ensured some internal validation by applying the methodology for design science research, we had validation of some chapters by peer-reviewing through the publication process and we implemented a tool that proves our MIPs are feasible, no validation step has been performed to assess the practical value of our framework for requirement engineers. By practical value, we mean that our models should bring them some interesting conclusions or advice that support the decision making process. Moreover, this value should be high enough to justify all the effort required to apply the current framework. The best way to validate the provided benefits is to apply our approach on a case study.

That being said, we believe that the present RP formulation at least brings new considerations and ideas in the field of RE by acknowledging the importance on solution comparison and optimization. This means that even if further validation research should highlight that some models, as they are currently formulated, do not bring expected returns, it would not necessary mean that requirements optimization problems are irrelevant.

In that case, we should maybe change some aspect of the models or apply another methodology. In no case, it should lead to a falsification of the present theory. Indeed, as suggested by Wauthelet et al. in [150], research in software engineering should be more based on an Lakatosian vision of science rather than a too radical Popperian perspective. This suggests that instead of abandoning the

present framework in case of inadequate validation results, we could still try to improve the framework (either the models or the methodology to use it).

4.2 Utility Function

Other limitations are related to the fact that our optimization models are based on utility functions. These functions are necessary to assess the value of nodes (revenues, costs...) that are then used in the objective functions or constraints. Using such function instead of preferences brings some advantages (discussed through the next chapters of this thesis) but have also different drawbacks. Here are the four main limitations we identified:

Evaluation: How to know utility value

The first limitation is concerned with the way the value of nodes would be assessed. Sometimes, this value can be quite easily estimated because there already exists some techniques or methodologies to evaluate it. We can take as example the evaluation of task effort in agile planning. In this context, the *Poker Planning* is used to estimate required efforts in terms of story points [71]. We show in Chapter 5 that the output of this method can be used with our approach.

However, many obstacles can appear in other situations when estimating the value of nodes is less straightforward. For instance, it might be really difficult to evaluate how much revenue can bring the satisfaction of the goal "*Customers can access visit results in a central repository*". In order to do that, we need to know how much the customers are ready to pay in order to get the goal satisfied. Answering this question is out of the scope of this thesis. Nonetheless, possible solutions could be found in techniques such as Analytical Hierarchy Process [126].

Cardinal Utility Functions

Secondly, using a cardinal utility function brings many interrogations. Among them is the question if we can consider that individual utility functions can be summed [70]. The question is relevant since many stakeholders could have different utility functions and consequently, we might not be able to sum utility values. Even if we can, we could ask if some stakeholders are more important than others? In this case, should we weight the stakeholder utilities? Or maybe the objective should rather be to maximize the minimal value obtained by the stakeholders instead of maximizing the aggregated utility function?

We think that those questions although being of prime importance are not an obstacle in the application of the present framework. Even if the models should be

transformed to apply one option rather than another one, it would be still possible to design such situations. In this thesis, we decided to restrain our maximization to an aggregated utility function. However, the reader should not forget that other maximization could be preferred but are still feasible. It would depends on the problem and could be seen as extension of our approach. We briefly discuss some potential further research regarding this problem in the conclusion (see Section 10.2).

Additive Utility Functions

A third main limitation regarding the utility function is that basic modeling of value-based goal models results in additive utility functions. An additive function is a function such that $f(a,b) = f(a) + f(b)$. In terms of nodes, it means that a solution satisfying goals G1 and G2 have the same utility that the sum of utility coming from those two goals considered individually. Formally:

$$u(\{G1, G2\}) = u(\{G1\}) + u(\{G2\})$$

However, it could be interesting to have non additive utility function. For example, the satisfaction of two goals could bring more utility because of some interactions. It is quite simple to illustrate: let us consider we are going to buy a car. A first requirement could be that the car is able to brake while a second requirement could be that it can accelerate. Although, each goal can have its own utility, the satisfaction of both would leverage more utility than the sum of individual satisfaction. Indeed, having a car that can brake but not accelerate have nearly no value. On the other hand, having a car that can both brake and accelerate have much more value that if the car could only accelerate. This situation is represented by:

$$u(\{G1, G2\}) > u(\{G1\}) + u(\{G2\})$$

It can be modeled in a goal model by inserting a third node which has the two others as premises. The utility of this node represents the marginal utility coming from the simultaneous satisfaction of G1 and G2.

For example, consider the situation depicted in Figure 4.1, where an individual has two goals: G1 *Watch high-resolution movies* and G2 *Watch movies on big screen*. Those goals could respectively leverage an utility of 5 and 15 units. The individual already owns a laptop equipped with a HD screen and a DVD reader. So, if he buys a Blu-ray player that he plugs on his laptop (assuming connectors are compatible) G1 will be satisfied. However, this purchasing will cost him 10 units of utility. Consequently, he prefers not buying the Blu-ray player: watching

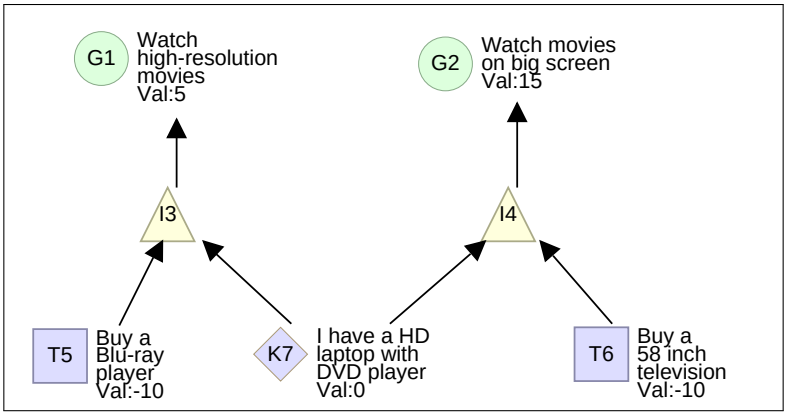


Figure 4.1: Example illustrating additivity property between utility of goal satisfaction.

HD movies on a small screen does not worth the 10 units of cost. Nonetheless, he can also buy a new 58 inch television in order to satisfy G2. If the television costs 10 units of utility, the individual will select this solution.

However, he feels that buying both Blu-ray player and the large screen would be even better than just the large television. This means that the simultaneous satisfaction of both goals has more value than the addition of individual goal satisfactions. In order to model this situation, depicted in Figure 4.2, we need to introduce an additional goal G0 which could be satisfied only if both sub-goals are satisfied. In this case, the three utility will be added. The total utility will be $10 (= 30 - 20)$.

Another situation is the decreasing marginal utility:

$$u(\{G1, G2\}) < u(\{G1\}) + u(\{G2\})$$

As an example, we can consider a set of requirements about a warehouse information system. The requirements state that a particular product in a large amount of many product references could be found by name, by description, by color and so on. We can easily understand that each new additional way to find the product bring marginally less value than the former one. Similar to the previous situation, it can be modeled by using additional nodes.

Although we showed that our utility function is not necessary additive, the method of inserting intermediate nodes is not scalable and still remains a limitation of our approach.

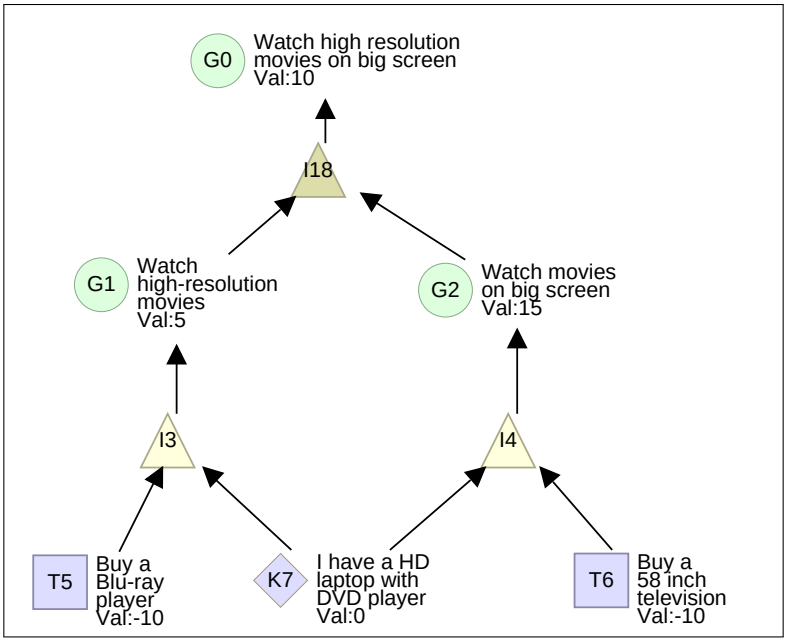


Figure 4.2: Example illustrating non-additivity property between utility of goal satisfaction by inserting a third *top*-goal.

Satisfaction as a binary variable

A final limitation is related to the satisfaction of softgoals. As defined in Chapter 3, softgoals are stakeholder desires that refer to conditions, the satisfaction of which is desired, vaguely constraint and not necessary directly mesurable. This last point, i.e. *not necessary mesurable*, makes them difficult to be integrated in our framework. The reason is that we use quantitative variables in the mathematical model while softgoals can explicitly be qualitative measures.

Nonetheless, it does not mean that they cannot be into account in our framework. We envisage two possible approaches.

1. Softgoals could be considered with a binary satisfaction. In that case, we should just find some sub-goals, tasks or any other concepts that could bring the softgoal in an acceptable level of satisfaction. Then, the goal model should be designed with those concepts as premises of the softgoals. Once the premises are satisfied, we consider the softgoal as satisfied as well. The limit of this approach is that it does not take into account the complete spectrum of softgoal satisfaction. It just defined a threshold from which we say

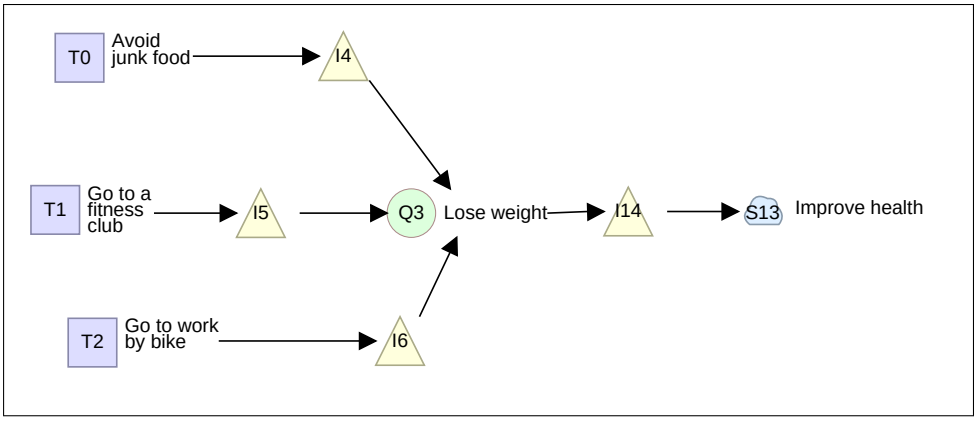


Figure 4.3: Goal model for the *Health Improvement* example.

Table 4.1: Task description for the *Health Improvement* example.

Task	Cost	Gain in weight
T0 *Avoid junk food*	-10	10kg
T1 *Go to a fitness club*	300	15kg
T2 *Go to work by bike*	10	10kg

that the softgoal is satisfied. This is the most simple integration of softgoals in our framework and this is the way it has been actually integrated.

2. We could also discretize the softgoal satisfaction and map each level to an utility value. Then with a quality constraint, we could map other concepts to the softgoal utility level. For example, let consider an individual who wants to improve his health. His goal model is depicted in Figure 4.3. He distinguishes three levels of improvement: *Highly improved*, *Improved* and *No improvement*. He is ready to pay respectively 250, 150 and 0 monetary units to achieve those states. We consider those amounts as his utility value. Then, a physician suggested him that if he loosed some weight, it will improved his health. Actually, he needs to lose more than 30kg to reach the *Highly improved* level, 20kg to reach the *Improved* and less than 20kg will not significantly affect his health. The individual identifies three tasks that can help him to lose weight. Their cost and the potential loosed weight are depicted in Table 4.1. The corresponding MIP is:

$$\begin{aligned}
\text{maximize} \quad &: \quad p - c && (4.1)\\
p &= 250s_H + 150s_M && (4.2)\\
1 &\geq s_H + s_M && (4.3)\\
s_H &\leq w/30 && (4.4)\\
s_M &\leq w/20 && (4.5)\\
c &= -10\sigma_{T0} + 300\sigma_{T1} + 15\sigma_{T2} && (4.6)\\
w &= 10\sigma_{T0} + 15\sigma_{T1} + 10\sigma_{T2} && (4.7)
\end{aligned}$$

where:

- p is the utility value resulted from the improved health,
- c is the cost to perform the tasks,
- s_H and s_M are binary variables representing the softgoal satisfaction level,
- w is the loosed weight,
- $\sigma_{T0}, \sigma_{T1}, \sigma_{T2}$ are binary variables representing the execution of the tasks.

These two suggestions are not aimed to be exhaustive. They simply suggest some potential integration of softgoals in our framework. Moreover, they are not incompatible to each other and could be used together if necessary. Nonetheless, we think that the second suggestion is probably the most adequate handling of softgoals although we limited this thesis to the first approach.

4.3 Computation Complexity and Scalability

Because our optimization problems are formulated as Mixed-Integer Problems, and due to their structure, they are close to the knapsack problem [54]. This latter is O^n complex which implies that the computation time goes exponentially with the number of decision variables (the graph size, the number of iterations...) This unfortunately means that our approach does not scale.

This limitation can however be balanced by using non-deterministic algorithms such as genetic algorithms [7]. Applying such methods as well as integrating them into AnalyticGraph.com is a possible track for further research.

That said, computation times are still acceptable with the basic **branch-and-bound** (B&B) algorithm for most goal models composed of maximum one hundred nodes. In order to estimate computation times in function of the node num-

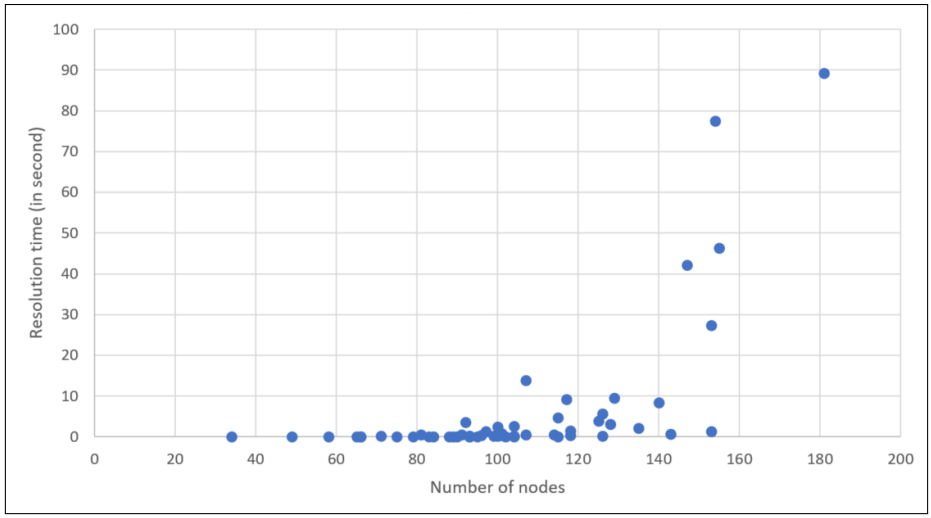


Figure 4.4: Resolution time in function of nodes number for 50 automatically generated goal models.

bers, a dummy model generator was developed [1]. Simulations are depicted in Figure 4.4. 150 nodes seems to be a turning point.

From an empirical basis, as we can expected, we identified that the resolving time depended, among other factors, on (1) the number of decision variables and (2) the density of the model. In this precise framework it has several implications.

For the first point, it is important to highlight that the number of decision variables does not only depends on the number of nodes. For example, if we introduce a time dimension (such as in Chapters 5 and 6), all nodes will have a specific decision variable for each period of time considered. For instance, a model with 50 nodes computed on 5 periods will consequently have at least 250 decision variables. Increasing the number of period could then rapidly threat the practicability of the approach. Consequently, our framework has the following empirical limit for model resolution with the branch-and-bound algorithm and no pre-processing optimizations :

$$nodes \times periods \leq 150$$

The second point is the density of the graph, i.e. the concentration of links between the nodes. In the context of Techne goal model, density is related to the

[1] The dummy model generator is accessible at `http://www.analyticgraph.com/dev/mod_opti/dummyConfigurator.html`

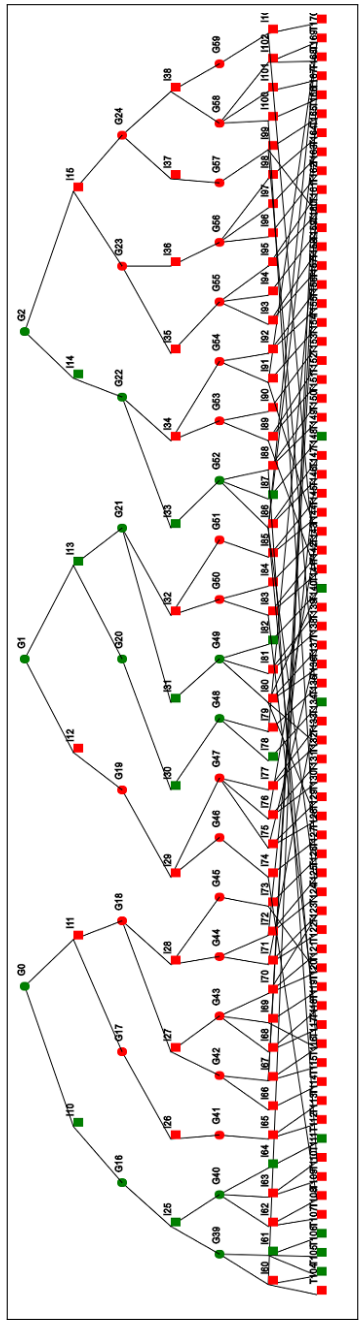


Figure 4.5: Example of a goal model resolved in 16.37 seconds

number of alternatives a goal can have (i.e. how many incoming links from inference node) and their own contribution to other node satisfaction (i.e. the number of outgoing links to inference a goal has). It has as consequence that a really sparse goal model consisting of 300 nodes can be resolved nearly instantaneously, while a very dense model will be no more computable after 80 nodes.

Finally, in order to help the reader to realize what means a goal model consisting of about 150 nodes, let consider the goal model depicted in Figure 4.5. It was resolved in 16.37 seconds and it is constituted of 167 nodes and 259 links. Top goals had a revenue varying from 0 to 300 units while the tasks (the bottom line of squares) had a cost varying from 0 to 30.

4.4 Considered and Required Information

One should be careful when using our approach. Eliciting, acquiring and evaluating exhaustively all possible requirements at the beginning of a project in order to compute the absolute optimal solution is impossible. The current approach is not intended to be used in such a way (as advocated by the waterfall methodology). The optimal solution our approach can compute has to be balanced with the available information at the moment of the optimization. In most cases, this information is unstable and incomplete. Consequently, we advocate to use this approach in an iterative way. The model can be run again when new information is acquired and integrated. This is why it is necessary to provide accessible and easy-to-use tools supporting our approach, making reevaluation of the model as easy as possible.

What information is available?

This perspective echoes Herbert Simon's bounded rationality [133]. Indeed, software engineering implies many stakeholders in the process of system development. Each stakeholder being nourished by beliefs, desires and assumptions, having time constraints, no one has a perfect reasoning. It is thus necessary to take into account that it is impossible to exhaustively study a problem by getting all information and process it ideally.

It means that our approach although trying to find the optimal solution to a requirement problem is not intended to find the *ideal* optimal solution. Since engineers cannot perfectly evaluate all the situations, rather than finding *the* best solution, they should seek to find the best perceived/expected solution.

To illustrate this discussion, let us consider a possible project that, in the best case, is believed to bring 30 units of utility while it is believed that it could bring nothing in the worst case. It is depicted in Figure 4.6. In the classical RE frame-

work, as soon as a satisfactory solution of the problem would have been identified, it would have been selected and used for the next steps of software engineering. In this thesis, we suggest that, at the early phases of the project, preliminary alternatives should be identified and evaluated. This is depicted in part (a) of the Figure. There, three potential solutions have been identified. Given some selection criteria, one optimal solution is selected (**B** in our example). As the project goes on, engineers discover that there are sub alternatives to **B**. After collecting more information on each alternatives, they decide to select the **B2** solution (which is a local optimum for **B**).

In a perfect world (i.e. with complete information), one would have selected the **C3** solution (see part (d) of Figure 4.6). However, at the early phases of the project, the available information did not let guess that the **C** path would have eventually been better. This illustrates that rather to reach the ideal optimal, our framework, because of the bounded rationality, can only claim for local optimum.

What amount of information do we want to collect?

From the previous discussion, it becomes interesting to wonder if the approach suggested through this thesis could be applied on short iterations of work, let consider 2 weeks?

We think that it could actually be applied in such context but it is necessary to provide guidelines on how to apply it. As an expressive example, consider the case of UML[2]. This specification language can be used very formally with many details and plenty of very specific concepts (e.g. qualified names, association navigability, isOrdered/isUnique attribute on multi-valued properties). Using UML like this, requires much more effort and probably a dedicated software because many details will not be graphically represented. However, although it will require much more time, it is interesting when we want to, for example, use it for code generation or when we want to end up with a detailed design allowing outsourcing. Nonetheless, UML can also be used in a more "sketching-way" in which it is used to discuss, suggest and communicate some ideas about a situation. In this type of use, the model is sketched on a white board in few minutes and requires much less effort. This is a more brainstorming fashion that can be used in short iteration projects.

The application of RO can also be distinguished in the same way. Depending on the purpose, the available time and the available information, the approach can be more elaborated and detailed or more lightweight. The latter case means that we will reduce one (or several) of these aspects:

[2] http://www.uml.org/

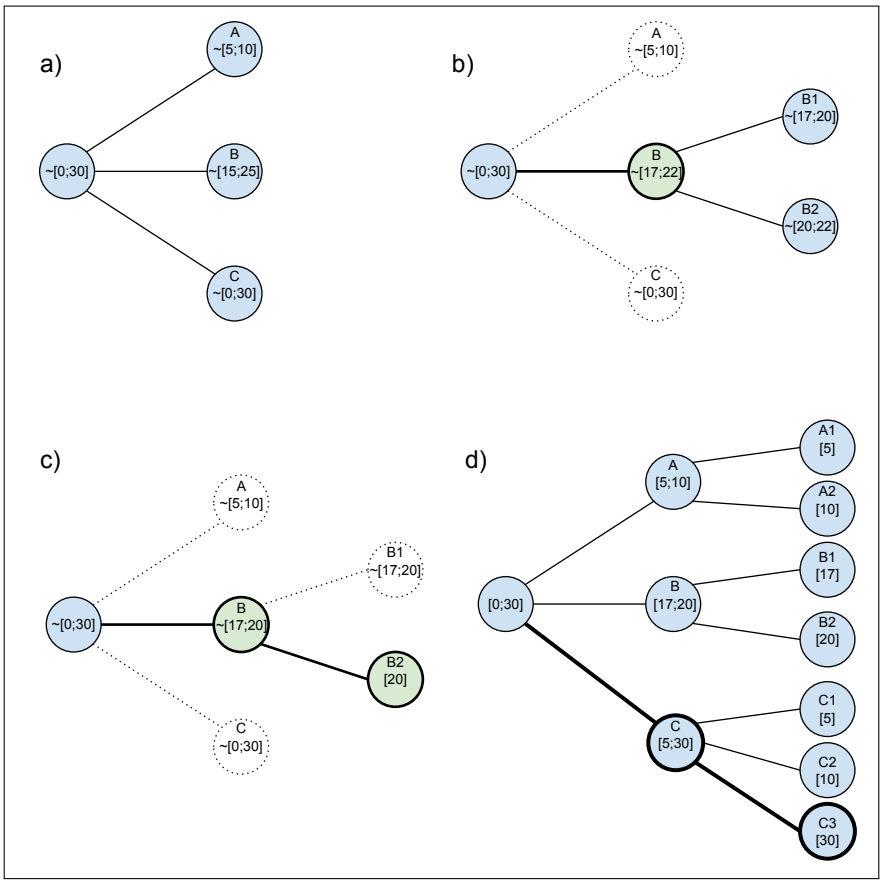


Figure 4.6: Illustration of three iterations (*a*,*b*,*c*) of use of the current framework when comparing alternative solutions under a bounded rationality hypothesis. Green circles are selected alternative. Numbers between brackets are worst case estimation (left) and best case estimation (right). A tilt represents an estimation. As information is collected, estimations are corrected. Case *d* is the perfect information situation (unreachable with the bounded rationality assumption).

- The depth of the goal model (i.e. how much details we want to provide regarding the refinement)
- The width of the goal model (i.e. the number of alternatives we are ready to consider)
- The precision of the estimation of the utility values

But even if the model is very simplified, we think that the current approach can be used for short iterations and bring some interesting thinking about the problem.

4.5 Implementation Limits

A final limitation is related to the current state of the tool. Some elements presented in this thesis have not yet been implemented. For example, we can cite the quality constraint concept such as presented in section 3.2. Moreover, defining a new language or a new Mixed-Integer Program still needs programming skills. A better solution should provide users with a graphical user interface designed to ease such creation. Although AnalyticGraph.com does not yet integrate those elements, the platform would be able to provide them in the near future.

Part II

Optimization Models

5 Optimization of Agile Planning

5.1 Forewords

This Chapter, the first among four chapters dedicated to specific optimization problems, tackles **RI4** which states *"How to determine which requirements have highest priority, for whom, and why?"* as well as **RI7** *"How to estimate costs, risks, and **deadlines** for making systems that satisfy requirements?"*. The choice of these particular Requirement Issues was quite obvious. First, priorities and planning estimation are considered as highly important issues in the literature (cf. the Next Release Problem) as well as in the industry (e.g. adoption of iterative methodologies). Secondly, it is the most recurrent optimization problem that MystShop faced.

We set our ROP formulation in the context of the *Agile methods*. Three motivations drove this choice: (1) this method family is gaining more and more popularity in software engineering what makes it a relevant and current problem, (2) as we discussed in the introduction, it is important to root Requirements Optimization in incremental and iterative environment, what agile methods are, (3) the similarity between user story refinement and goal refinement makes agile method the best candidate for this chapter. Results of this chapter has been presented in:

J. Gillain, I. Jureta, and F. Stéphane. Planning optimal agile releases via requirements optimization. In *Third International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'16)*. Springer, 2016

5.2 Introduction

Context: Requirements Priority in Agile Projects

Agile methods have become popular for organizing software development. Many benefits have been touted [116]: better knowledge transfer, active participation of the customer in the project, incremental deliveries, and so on.

Relative to more established methodologies (e.g., waterfall), which are based on the assumption that the problem is fully specifiable and a solution can be en-

tirely predicted [26], agile methods are based on the assumption that customers, users, stakeholders in general are not able to specify all requirements of the system-to-be at the beginning of the software process, due to requirement or environment change, lack of experience in the problem domain, etc. [114].

Agile tries to treat requirements continuously through the whole project life cycle. Development is iterative, the development team periodically delivers increments of functionality by focusing on requirements with highest priority at each release planning. It ensures that value comes continuously during the complete process and stakeholders are then being able to test and validate the highest priority requirements. Those tests helping the customer clarify and refine subsequent requirements, for later releases.

To make such an approach working for customers, requirements prioritization is important. The Agile Requirements Problem (ARP) is driven by elicitation of *User Stories*. It starts with the definition of high-level statements, *epics*, written from the system user's perspective, then refined into shorter, more narrowly scoped user stories. This refinement process is ongoing, running throughout the system engineering process. The literature suggests that the priority of each user story should be given by the customer to the *Product Owner*.

In practice, it is known that other criteria (than the value for the customer) enter into account when prioritizing user stories. They can be, for instance, the effort estimated to implement each user story, or the functional dependencies [8]. It seems important that an effective method for ARP solving should take those additional criteria into account.

Scope: Goal Models Agile Requirement Problems

Agile requirement methods and tools have often been opposed to traditional requirements methods [114]. However, it also has been demonstrated that mappings between agile concepts and traditional requirement engineering methods are possible, especially with goal modeling (GM) [148]. As suggested by Wautelet et al. in [149] agile planning could benefit from the use of goal models which could improve the consistency of the sets of User Stories to be implemented. We also think that such models could be used to prioritize those sets of User Stories.

As described in [148], the three dimensions are traditionally suggested in User Stories (i.e. the WHO, the WHAT and the WHY) finds their corollary in goal modeling. Moreover, similarly to user stories, goal models are based on a refinement process in which abstract goals are refined into more concrete ones. In both cases (goal modeling and user story writing), the refinement process ends up with an operationalization into tasks. Another common point is that both user stories and goals are desired states the system should satisfy, or bring about.

However, goal models differ from user stories since they allow among others to explore alternative solutions or to assess conflicts between goals. They also support modeling of non-functional requirements while it is not obvious how such requirements should be incorporated into user stories[1].

The variability of goal models (i.e. the exploration of alternative solutions) seems truly interesting for agile projects since they do not define the whole product in the early stages. They instead specify a *vision*, which is imprecise and abstract, and gives thereby a space with potentially many alternative solutions (functionalities) that the system could implement [130]. Variability is even more interesting when considering that a first "fast and cheap" version of the system could be delivered, before investing in a more comprehensive solution to be brought to market via subsequent releases.

Problem: Which requirements to satisfy when?

As we just discussed, the Agile Requirements Problem is to provide first the highest valuable requirements on the basis of business value. It involves the following sub-problems:

Evaluation problem: How to assess this business value because it is not independent of the implementation cost. As an example, consider 2 candidate user stories for the next release of a system, let's say a and b (with respective business value of \$8000 and \$5000). Before deciding which one should be selected, it seems reasonable to also take into account their relative cost. If a costs \$5000 to be implemented while b costs \$2000, it is clear that b should be implemented first. Then, business value should be balanced with effort estimation.

Selection problem: A second problem comes from the difficulty to select which alternative should be implemented when facing a lot of variability in requirements. The customer has to select between potentially dozens of combination of features. This is very difficult and time consuming without a decision support tool.

Release problem: A third problem is to determine in which order the selected solution should be delivered. It is about deciding a release planning. The main criteria being to provide the most value as soon as feasible in the life cycle.

Contribution: An Optimization Problem and its Resolution

In this chapter, we suggest how to model ARP with goal models and provides a mapping to a Mixed-Integer Program rooted in CORE. The solutions of this

[1] One can cite Alistair Cockburn's suggestion to formulate them as constraints [36]. This approach can be transposed to goal models

MIP automatically define a planning release taking into account business value, functional constraints and required effort. We also present the implementation of the approach in AnalyticGraph.com.

The chapter is structured as follows. First, we show in section 5.3 how to model ARP instances with goal models. Then, in section 5.4, we present the mapping between the goal model and a mixed-integer program. Section 5.5 presents the solution of the MIP. We present features developed in AnalyticGraph.com in order to support this approach in section 5.6. Eventually, we discuss related work and we conclude respectively in sections 5.7 and 5.8.

5.3 Modeling Agile Requirements with Techne

In this section, we present how we model ARP instances with Techne. We first discuss the goal refinement process and relates it to the refinement of user stories. Then, we show how to take into account of the business value and task effort in the goal model.

Refinement

In order to illustrate the refinement process of user stories with a goal model, let's consider the goal model of MystShop we introduced in the previous chapter in Fig. 3.2. All the requirements have been reformulated as user stories. Actually, this mapping between goal modeling concepts and user story concepts is far being simply a rephrasal as we did between the two models [149]. However this process is out of the scope of this thesis.

As we discussed in Chapter 2, MystShop sends out mystery shoppers, that is, individuals unknown at a shop, to make a purchase and subsequently evaluate the purchasing experience. This modified version of the goal model 3.2 fitting User Story redaction prescripts [37] is depicted in Fig. 5.1. This is the model that will be used in this chapter.

There are three abstract user stories (i.e *epics*):

- G0: As a Mystery Shopper (MS), I want to make visit results available to the customer,
- G16: As a Customer[2], I can access visit results in a central repository,
- G37: As a Customer, I can explore data through visualizations

[2]By *Customer*, it is understood the Mystery Shopper company's customers.

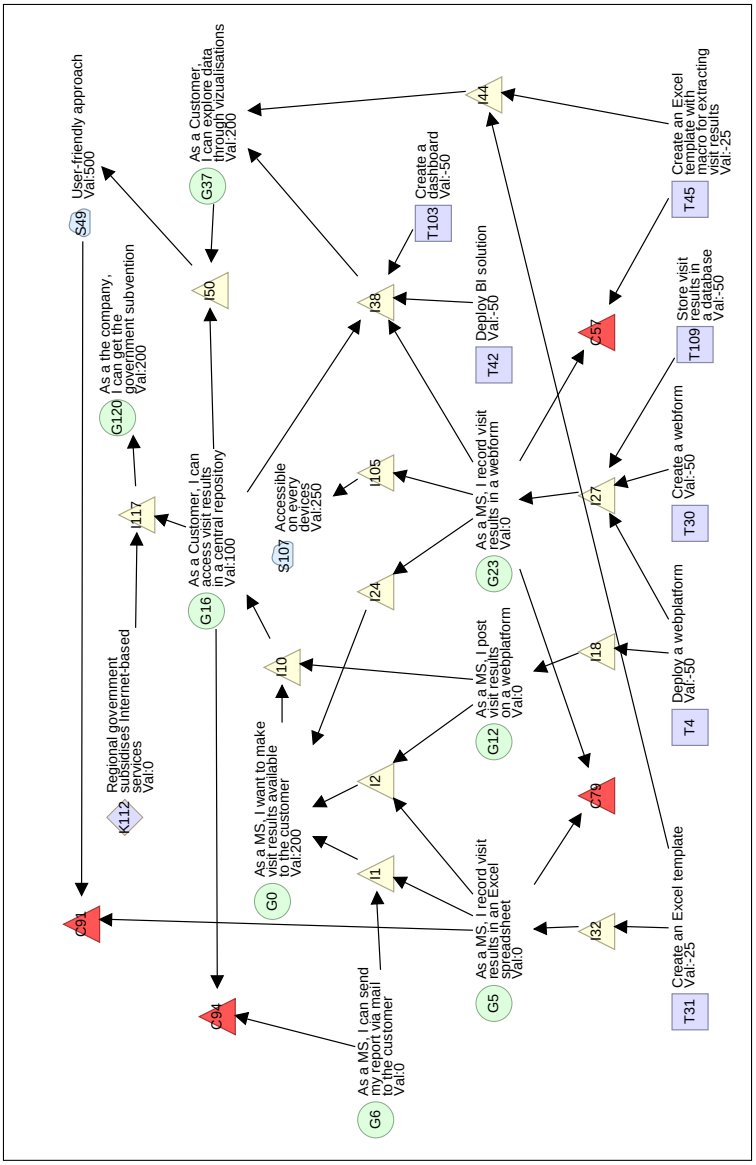


Figure 5.1: Modeling of the MystShop Case. Accessible and editable at `http://analyticgraph.com/dev/?g=R2CinFhvUK`

Each of these user stories are depicted as a goal in the goal model. However, to make them practical for development, they need to be refined into more concrete ones. For example, we could consider that G0 will be satisfied if a mystery shopper can record visit results in a spreadsheet (G5) and then he can send it via mail to the customer (G6). This refinement can be depicted in the goal model. In Techne, we use an inference relation node (I1 in Fig. 5.1) between G0, G5 and G6. This relation says that

$$I1 : G5 \wedge G6 \rightarrow G0. \tag{5.1}$$

It is also possible to model some more functional dependencies between goals. For example, we cannot consider that G16 is satisfied if G0 has not been. This is done by adding G0 in the premises of all inference relationships which would conclude G16 as satisfied.

In an agile methodology (e.g. SCRUM, XP), once user stories have been refined to an adequate level of granularity, we can ask developers to define underlying tasks to be done [104]. Those tasks would integrate the Sprint Backlog, i.e. a set of tasks to be done during the next implementation iteration. In our goal model, this is achieved by operationalizing goals into tasks. For example, G5 could be considered as "done" if developers create an Excel template that will be used by mystery shoppers (T31).

This approach fits a SCRUM approach [131]. There, a Product Backlog (PB) is created and iteratively filled. Each item of the PB is a user story that can be refined more and more as the project progresses. User stories are always a description of a functionality of the system-to-be from a user perspective. In other words, it is a desired state that the user would like. It is similar to the goal concept in goal modeling. Once a user story is selected for the next sprint, developers filled a Sprint Backlog with the underlying tasks for a given amount of effort.

The effort required to implement those tasks (or user stories not yet refined into tasks) are estimated in *story points*. For this, one can use poker planning [104]. It is however necessary to take into account these story points in our goal model.

Business Value and Story Points as Node Label

Another important concept in agile processes is requirements prioritization. Requirements (user stories) with the highest priority would be implemented first. The agile literature insists on the fact that the business value should be the first driver for prioritizing. However, in practice we can see that it is not the only element to be considered for prioritizing [116]. Another important criterion to consider when prioritizing is the required effort, which determines development cost.

Both business value and cost can be integrated in our goal model. It takes the form of a label which is associated to each node (goal or task). In the MystShop example depicted in Fig. 5.1, values are depicted as node labels prefixed by Val. For example, the goal G0 will leverage 200 units of value if it is satisfied, while T31 will consume 25 units of resources (it could be story points).

When speaking about revenue acquired from goals, it is important to distinguish two types of revenue: *recurrent revenue* and *unique revenue*.

In the case of the recurrent revenue, the achieved goal will yield revenue continuously as long as it is satisfied. For instance, consider goal G37 stating that a "Customer can explore data through visualizations". Once this goal is achieved, MystShop will asks additional $10 per visits since the service offered has higher value. In this case, this goal has a recurrent revenue. For this type of goal, the earlier they are satisfied, the better it is.

With unique revenue, we model goal which yields a single amount of value. For example, we discussed in Chapter 2 that MystShop had the possibility to get a subvention from government as soon as the company had an Internet platform (G120). However, it can be accorded only once when goal G16 is satisfied and under the assumption (K112) that the government effectively subsidies it.

Regarding task implementation, some tasks need to be done only once and can be considered as definitely done. We call them *persistent task*. Others do not have this property. For instance, a maintenance task. It becomes then interesting to consider that a goal could be satisfied either by a cheaper but repetitive task or by a more expensive task that automatize the repetitive task.

Both recurrent revenues and persistence tasks are boolean properties associated with nodes in our goal model.

Conflicts between goals

Conflict management is important in this approach, because the solution should not produce a release planning whose increments are individually conflict-free but the integrated solution presents several conflicts. It implies that the conflict needs to be checked at each iteration.

Nonetheless, it can be interesting to deliver a first version of a system providing some features and removing them when a next release introduces more elaborated features (conflicting with the previous release).

In our example, we know that if mystery shoppers record their visits in a web-based form it will be conflicting with recording them in a spreadsheet. This is depicted with conflict C79. Moreover, the spreadsheet macro would be useless to preparing visualization if data are no more recorded in a spreadsheet (conflict C57).

Another interesting conflict is that if customer can access visit results on a central repository and that some analytics are deployed, the solution should be more user-friendly than simple excel reports. Of course, it requires than no more reports should be registered in spreadsheet (conflict C91).

Eventually, the central repository would be useful iff mystery shopper stop sending their report via mail (conflict C94).

Project progress

An important aspect of agile projects is that requirements are expected to evolve through the whole project. One should be able to modify the goal model in the same way the product backlog evolves.

After a particular sprint has ended, one should change the story point of a task to 0 if that sprint completed the task. If only part of the task is completed, task effort can be reduced proportionally to the work done.

For new goals or tasks that are identified throughout the project, they can be integrated later and the model can be reevaluated.

5.4 Mathematical Model

This section presents our mapping between the ARP modeled with Techne and a Mixed-Integer Program that extends the basic MIP model 3.1. The complete program is given in Tab. 5.1 and Tab. 5.2. Here after, we discuss in more details specific modifications brought to the basic MIP presented in Chapter 3. One of the most fundamental change is the introduction of a multi-period reasoning able to take into account iterative development of agile projects (namely the *sprints* in Scrum).

Objective function

Priority in agile methods should be to maximize business value delivered to the customer. If we define u as an *utility function* mapping each goal to a business value at each sprint, we can define the objective function as:

$$\max \sum_{p \in P} \sum_{r \in R} \mathsf{u}(r,p) * \sigma_{r,p}$$

where:

- P is the set of sprints $\{1,\ldots,p\}$,

- R is the set of goal model nodes (goals, softgoals, quality constraints and domain assumptions),
- $\sigma_{r,p}$ is a binary decision variable setting if the goal r has been achieved for period p (equals to 1 iff the node is satisfied, 0 otherwise),
- $\mathsf{u}(r,p)$ is a function relating a node r to a specific utility value for the given sprint p.

For modeling the importance of delivering features as soon as possible (in other words, the urgency), a discount rate can be applied on the utility function, resulting in a situation where $\mathsf{u}(r,i) > \mathsf{u}(r,j)$ if $i < j$ with i and j being the sprint number. In the same idea, we can also model a window of opportunity for some features, i.e. some features will have no more value if delivered after a particular deadline.

MIP 5.1. First part of the MIP description for ARP.

Constants		
v		Velocity
s		Sprint numbers
Sets		
P	$= \{1, \ldots, s\}$	Iterations (Sprints)
N	$= G \cup T \cup S \cup Q \cup K$	Concept nodes (goals, tasks, softgoals, quality constaints and domain assumptions)
G^*	$\subseteq G$	Unique revenue goals
$\overline{T}$	$\subseteq T$	Persistent tasks
$\widetilde{T}$	$= T \setminus \overline{T}$	Non-persistent tasks
M	$\subseteq N$	Mandatory nodes
C	$= \{c_0, \ldots, c_i\}$	Conflict nodes
I	$= \{i_0, \ldots, i_i\}$	Inference nodes
N^*	$= N \cup I$	Graph nodes
R	$= N \setminus T$	Revenues nodes
Decision Variables		
$\sigma_{N^*,P}$	$= \{\sigma_{i,p} \in \mathbb{B} : i \in N^*, p \in P\}$	Binary variables representing node satisfaction.
$\alpha_{\overline{T},P}$	$= \{\alpha_{i,p} \in \mathbb{B} : i \in \overline{T}, p \in P\}$	Binary variables representing satisfaction of persistent tasks.
Functions		
u	$: N \times P \to \mathbb{R}$	Utility function
$\mathsf{in}(x)$	$\subseteq N \cup I$	Incoming nodes function
$\mathsf{in}_Y(x)$	$= \mathsf{in}(x) \cap Y$	Typed incoming nodes function

MIP 5.2. Second part of the MIP description for ARP.

Objective function

$$\max \sum_{p \in P} \sum_{r \in R} \mathsf{u}(r,p) * \sigma_{r,p}$$

Constraints

Premises constraints
$\forall p \in P, \forall i \in I$:
$$\sigma_{i,p} \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_{x,p}}{|\mathsf{in}(i)|}$$

Conclusion constraints
$\forall p \in P, \forall n \in \{x \in N : |\mathsf{in}_I(x)| > 0\}$:
$$\sigma_{n,p} \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_{i,p}$$

Persistent task realization
$\forall p \in P, \forall i \in \overline{T}$:
$$\sigma_{i,p} \leq \sum_{q \in \{1 \ldots p\}} \alpha_{i,q}$$

Unique revenue constraints
$\forall g \in G^*$:
$$1 \geq \sum_{p \in P} \sigma_{g,p}$$

Conflict constraints
$\forall p \in P, \forall c \in C$:
$$\sum_{x \in \mathsf{in}(c)} \sigma_{x,p} \leq 1$$

Mandatory constraints
$\forall i \in M$:
$$1 \leq \sum_{p \in P} \sigma_{i,p}$$

Velocity constraints
$\forall p \in P$
$$v \geq \sum_{t \in \overline{T}} \mathsf{u}(t) * \sigma_{t,p} + \sum_{t \in \overline{T}} \mathsf{u}(t) * \alpha_{t,p}$$

Constraints for Value Persistence

As discussed in section 5.3, their are different types of goals and tasks regarding how their business value/cost should be considered. Goals can bring recurrent revenues as long as they are considered as satisfied or on the contrary, they can bring a unique revenue even if their satisfaction last several sprints. By default, the model supports recurrent revenue. An additional constraint set is needed to model single revenue goal, we call it *Unique Revenue Constraints*.

Regarding task persistence, some tasks need to be done each time it is necessary to use them, while other tasks are required to be executed once. Default behavior of the MIP is the non persistent tasks while an additional set of constraints is required for persistent tasks.

Unique Revenue Constraints

Ensuring unique revenue of some goals is done with:

$$\forall g \in G^* : 1 \geq \sum_{p \in P} \sigma_{g,p}$$

where G^* being the set of goals with unique revenue.

Persistent Task Realization

For persistent task, we need to specify that if the task has been developed during a previous iteration, it can be considered as satisfied for the following iterations. It is done with the following constraints:

$$\forall p \in P, \forall i \in \overline{T} : \sigma_{i,p} \leq \sum_{q \in \{1 \dots p\}} \alpha_{i,q}$$

where:

- $\overline{T}$ is the set of persistent task
- $\alpha_{i,q}$ is a binary decision variable assessing that the task i has been realized during sprint q.

The constraint is read as following, the task i can be considered as satisfied for period p (i.e. $\sigma_{i,p}$ is set to one) iff it has been realized during the current or a previous sprint.

Velocity Constraint

Agile methods work iteratively. For example, a SCRUM project is divided into *Sprints* (i.e. an duration-fixed iteration of work) in which developers have to develop a certain amount of story points. This amount is called the team velocity.

Adding this aspect into the MIP is done by adding a new set of constraints called hereafter *velocity constraints*. If we define v as the team's velocity and P as the set of all sprints in the project, we formalize the constraints as:

$$\forall p \in P : v \geq \sum_{t \in \widetilde{T}} u(t) * \sigma_{t,p} \sum_{t \in \overline{T}} u(t) * \alpha_{t,p}$$

where:

- $\overline{T}$ is the set of persistent tasks
- $\widetilde{T}$ is the set of non-persistent tasks

Mandatory Constraint

In comparison with the basic MIP 3.1, mandatory nodes have to be handle differently. Indeed, since the optimization is computed on several iterations, mandatory requirements are not necessarily satisfied for the first iteration. Consequently, we just want to ensure that they have been satisfied once during the whole process.

$$\forall i \in M : 1 \leq \sum_{p \in P} \sigma_{i,p}$$

5.5 MIP Solution

The provided solution of the mathematical program states in which sprint should a task be planned and it also describes how to achieve each goal in each sprint. It is depicted in Table 5.1.

After the first sprint, MystShop will get a first release in which employees would be able to use an Excel template in order to send results by mail to the customers. This Excel would integrate some analytics allowing the customer to visually explore the results. After the second sprint, a web-platform would be deployed, enabling MystShop employees to publish Excel spreadsheet on it and stopping the use of mails. Having a web-based application should allow MystShop to get the government subvention. There is no release after sprint 3 which is dedicated to create a web-based form implementing the checklist. After the sprint

Table 5.1: Solution of mystery shopper case

		Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Tasks	Story Points						
T31 Excel templ.	25	x					
T45 Excel reporting	25	x					
T4 Deploy web-platform	50		x				
T30 Create a webform	50			x			
T109 Store results in a db	50				x		
T42 Deploy BI Solution	50					x	
T103 Create a dashboard	50						x
	SP size	50	50	50	50	50	50
Goals	Revenue						
G0 Register visit	200	x	x	x	x	x	x
G5 Insert xls	0	x	x	x			
G6 Send by mail	0	x					
G12 Post on web	0		x	x	x	x	x
G16 Register central	100		x	x	x	x	x
G23 Insert web	0				x	x	x
G37 Xplore data	200	x	x	x			x
G120 Get subvention	200		x				
S49 User-friendly	500						x
S107 Access. on many devices	250				x	x	x
	Revenue	400	700	500	550	550	1250
	Cumul. Rev.	400	1100	1600	2150	2700	3950

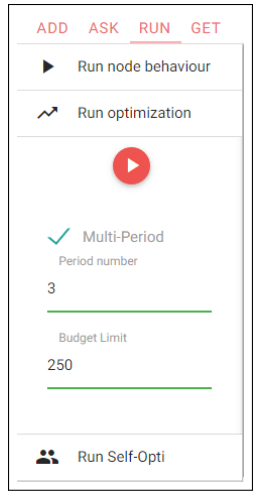


Figure 5.2: Parameters for Agile Planning Optimization in AnalyticGraph.com

4, where developers should work on storing in a database information acquired from the form, MystShop employees should give up Excel spreadsheets in order to start working with the web-based application. However, since there is no specific analytics developed for this application, visual exploration of data will not be possible until sprint 6.

An interesting conclusion from this planning is that it would be more profitable for MystShop to have a first solution using Excel files while another more integrated solution would come in next sprints. This migration would result in two periods (after sprints 4 and 5) where the customer will have no more access to data exploration because of the migration. Nonetheless, it is still the planning maximizing its global satisfaction.

5.6 AnalyticGraph as Supporting Tool

All previously presented models and modeling primitives are accessible on AnalyticGraph. Specific aspects related to optimization of ARP are supported by the tool (node values, goal revenue recurrence and task persistence). It is also possible to specify the number of sprints to be considered as well as the velocity constant 5.2. Results of the optimization are presented both graphically and as a list in AnalyticGraph.com. The case presented in Fig. 5.1 can be accessed at `http://analyticgraph.com/dev/?g=R2CinFhvUK`.

5.7 Related Work

The issue discussed in this chapter is on the boarder of two research disciplines. First, it deals with requirements optimization since it tries to identified an optimzation set of requirements between different alternatives. Secondly, it is directly related with what is called the Next Release Problem [7].

Several approaches have already been suggested for applying optimization techniques on requirements engineering. In their paper, Zhang et al. discussed them in general by presenting advantages and challenges [157].

One of the first application is suggested by Jung [84]. It gives developers a method to balance the cost and value of the requirements, and then implement the most cost-effective set.

Bagnall et al. were the firsts to coined the term *Next Release Problem* [7]. It is about selecting a set of requirements that is deliverable within a budget and which meets the demands of the customers. Our approach goes a step further by providing a mapping between an actual requirement modeling language (Techne) to a MIP. We also further elaborate the notion of customer satisfaction by distinguishing two types of revenues.

In their work, Zhang et al. explored the multi-objective next release problem by distinguishing the revenue maximization and the cost minimization [158]. Their work mainly consists of comparing search techniques for multi-objective optimization problems. Although separating cost and customer values seems interesting, our approach can be applied with a profit function (i.e. a fitness function summarizing revenue maximization and cost minimization) [57]. Moreover, our work is less focused on comparing search algorithms but rather on providing a mapping between goal modeling and MIP.

In their work, Ruhe et al. suggested an approach mixing what they call the *art of release planning* and the *science of release planning* [125]. It results in a model taking into account dependencies between features, resource constraints, urgency of features and different stakeholders point of view. Our approach differs from theirs because there is no attempt to balance multiple stakeholders satisfaction and urgency rates. However, urgency is managed with diminishing values of goals through time.

Saliu et al. focused on release planning optimization for evolving systems [127]. They proposed a new release planning framework that considers the effect of existing system characteristics on release planning decisions.

In comparison with all previously mentioned work, our approach is more focused on providing a mapping between an existing requirement modeling language than a study on performance of various algorithms. They should then be considered as complementary to ours.

5.8 Conclusion

This first formalization of a ROP suggested a method able to support optimization of release planning in agile projects. It directly relates to **RI4** from Tab. 1.2. A first contribution was to suggest modeling of the Agile Requirement Problem with goal models. A second contribution was a mapping between the goal model and a Mixed-Intger Program. Eventually, we briefly presented an implementation of the approach on AnalyticGraph.com.

Although our optimization problem focuses on business value delivery (as agile principles prescribes), it also takes into account implementation effort and feature dependencies (through inference relationships).

The approach allows each user to focus on their domain, the product owner defines a product backlog in the form of a goal model (which replaces the traditional list of user stories), developers identify implementation tasks to be done during sprints and evaluate their required efforts with story points. Then the release planning (and consequently priorities) is computed by resolution of the mixed-integer program.

In summary, this chapter contributes to our three research questions since (1) it showed that **RI4** could be relevantly formalized as a ROP in CORE, (2) we described some extensions of Techne with labels on nodes for satisfaction persistence, and (3) we described the mapping with a MIP.

6 Optimization of a Software Product Line Portfolio

6.1 Forewords

This second chapter of Part II tackles **RI14** which states *"How to do RE for systems that should adapt to different environments and requirements?"*. The underlying problem behind this formulation relates directly to reusability. This problem was faced in the MystShop case when the possibility to reuse what have been done for MystShop could be reused for other purposes. This problem has been the main focus of the Software Product Line Community [35, 117]. This is why we set this optimization problem in the Software Product Line context.

Results of this chapter has been presented in:

J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck. Product portfolio scope optimization based on features and goals. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 161–170. ACM, 2012

6.2 Introduction

Context: Software Product Line

A Software Product Line (SPL) is defined as *a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way* [35]. Even if it provides customized software products, Software Product Line Engineering (SPLE) is currently regarded as an efficient approach to achieve large scale reuse. By adopting SPLE, an organization can help achieve different objectives, e.g., reduce their cost, provide adapted solutions for a variety of customers or decrease the time-to-market of software products.

Scope: Product Line Portfolio Scoping

When developing a SPL, one of the main factors impacting the previously cited objectives is the product line scope. As described by P. Clements in [34], scoping

is defined as the " *activity that bounds a system or set of systems by defining those behaviors or aspects that are in and those behaviors or aspects that are out*".

According to Schmid [128], there are three types of scoping. *Product portfolio scoping* aims at identifying the particular products that should be developed as well as the features they should provide. *Domain scoping* is the task of bounding the domains that are relevant to the product line. *Assets scoping* aims at identifying functional parts of the product line that should be developed in a reusable manner. Three goal levels for each scoping type can be distinguished: *identification*, *evaluation* and *optimization* of scope. As highlighted by recent reviews, current scoping methods fail in providing optimization methods. Some provide optimization for the product portfolio scope but there are no domain or feature optimization methods yet [106][79].

Problem: Finding the mix between commonality and variability

Determining an optimal scope is mainly a trade off question. On the one hand, if the scope is too large (i.e. the SPL includes many products which cover a large set of different markets), product members vary too much and the commonality is reduced. Consequently, economies of scales eventually drop and the time-to-market reduction is not achieved. On the other hand, if the scope is too narrow, the core asset base (i.e. the common part) does not satisfy the needs of enough customers and the return on investment never materializes. It is then necessary to carrefully define the scope in such a way that the optimal balance between commonality and variability is leveraged.

Contribution: A MIP to optimize software product portfolio scope

In this chapter, we present a mathematical model based on the joint use of goals and features. We show how this approach can help when optimizing the product line scope, especially product portfolio and assets scoping. We suggest that when scoping, identification and evaluation of the three types of scoping can be performed separately. Nonetheless, finding a global optimum for the product line scope requires that the optimization of the three scoping types are performed in an integrated model. Deciding which features will be included in each product, as well as determining which features will be reusable or choosing domains can not be executed independently. Otherwise the optimization would likely return a local optimum.

In order to integrate those three considerations into a single model we decided to use both goals and features in a product line life cycle profit optimization. The reasoning behind such an approach is that the final purpose of all product line

objectives is to deliver value. Moreover, we assume customers' Willingness-to-pay (WTP) for a system-to-be is function of the utility it provides and this utility is function of the satisfied goals and not of the provided features. Indeed, some features have value only when used with others, e.g., because of feature interactions. Goal models are thus used as a problem-oriented approach while features are solution-oriented aspects and the proposed method of scoping optimizes the matching between both aspects. This approach has to be considered as a complementary method with current approaches such as PuLSE-Eco [128].

In the next section, we present the idea of using both goals and features with illustration on the MystShop case. We also discuss how profit maximization objective has consequences on the other SPL objectives. The mathematical model is presented in section 6.4. In section 6.5, we explain why further commonality and variability analysis are required after the execution of the MIP. The adaptation of the model to different development contexts is described in section 6.6. We then illustrate the applicability of our model by applying it on an extension of MystShop case study. We finally discuss related work and conclude this chapter in Sect. 6.10.

6.3 Features and Goals for Scope Optimization

As highlighted by K. Schmid in [128] and D. Nazareth et al. in [110], advantages resulting from software reuse in SPLE can take several forms. It is important to realize that such advantages are mainly raised by economic considerations rather than technical factors. These potential reuse benefits are:

- reduction in development cost,
- reduction in time-to-market,
- increase in programmer productivity,
- improvement in software quality,
- improvement in maintainability,
- improvement in project planning.

Determining the scope of the product portfolio will have an impact on those benefits. However, assessing and predicting those benefits are difficult since varying the scope can have different effects on those benefits. Determining an optimal scope regarding all those advantages could require the use of a multiple objective decision framework. However, such process raises several difficulties such as expressing the preferences between those objectives.

Our method is based on the assumption that all the above objectives are driven by the same general purpose: profit maximization. For example, reducing the time-to-market aims at getting revenue earlier and benefit from market opportunities. Considering only this single objective allows to design a mathematical model taking into account the previous cited considerations. Generally speaking, the software product line profit is the difference between the revenue and the cost of this product line. The scope of the product line is an important factor influencing profit. By increasing the scope, an organization can increase its revenue since the set of potential customers will increase due to a larger set of products. However, as discussed in section 6.2, it will increase the variability and decrease the commonality between product members implying a reduction of the economies of scale on the side of the development cost. As discussed above, identification and evaluation of the product line scopes have to be performed before optimization. Regarding our model, *identification*, the first step of scoping, will result in a set of goal models for different customers or domains, a set of relevant features to be included in the future products and an identification of reusability of these features. The second step, namely the *evaluation* phase, will consist in assessing the value of goals, costs of features and costs of reuse. The final step is the *optimization* of the SPL scope on the basis of the previously identified and evaluated artifacts. We briefly discuss the identification and evaluation steps in this section but a complete description of those first two steps is out of the scope of this approach. We will focus on the third step of scoping in the next sections of this chapter.

Software Product Line Revenue

In order to generate revenue (e.g. by being sold), products from a SPL have to satisfy requirements of customers. Before deciding which product will be part of the SPL, we need to know what are the requirements of potential customers. In this context, goal models present two advantages in comparison to other requirement methods such as object-oriented requirement modeling. First, it allows us to analyze alternative ways to satisfy same requirements. E.g., two customers from different domain can have several alternative solutions satisfying their requirements, two of which are satisfiable with a common SPL. Comparing alternative solutions allows to find the largest possible core assets base for the product line to-be. Secondly, goal models capture the purpose of the software which is likely to have some stability. It does not address how it will achieve that purpose. Those requirements are more stable which is important because determining the scope need to be performed on quite stable considerations. Indeed, it is more difficult to modify the scope than a feature.

Moreover, as previously discussed, goals are good utility indicators. Cus-

tomers are willing to pay for a system which satisfies their goals, as this is how the client can understand the value of the system-to-be, rather than consider the specific features. For instance, mystery shoppers of MystShop do not care if they fill data in a webform or in a spreadsheet. What really matters is the value that can be leveraged from the use of this feature (e.g. storing data in a database make them available for other application). As highlighted in [106], methods such as the conjoint analysis can be used to evaluate the willingness to pay of a customer. However, further considerations about the identification process of customer needs and the method to be used in order to evaluate them is out of the scope of our research questions. What interests us in this thesis is that the requirement problem for SPL can be stated as follows: finding a set of tasks (i.e., ways of achieving goals) such that under some domain assumptions, stakeholders goals are satisfied [89].

As already discussed in the introduction of this thesis, using the CORE ontology for requirement engineering [89], the requirement problem can be formalized as follows $K, T \vdash G, S, Q$, where K, T, G, S, Q are respectively sets of domain assumption, tasks, goals, softgoals and quality constraints. Given a requirement problem, SPL engineers are interested in finding a product (i.e. a valid set of features) able to realize tasks of the requirements problem, formally $K', p \vdash T$ where K' states the conditions of tasks realization by the product and $p \in \mathcal{P}(F)$[1]. For instance, if we want to realize task t and we know that $f \rightarrow t$ then, a possible product could be the single feature $\{f\}$. In this trivial example, $K' = \{f \rightarrow t\}$. Consequently, the extended requirement problem for a particular customer becomes $K'', p \vdash G, S, Q$ where $K \cup K' = K''$ Then, the process of developing a product line consist in finding a $PL = \{p_1, \ldots, p_n\}$ such that for a given set of n customers with $i = (0 \ldots n)$, there is a product $p_i \in PL$ such that $K_i'', p_i \vdash G_i, S_i, Q_i$ with p_i being a product member of the SPL.

Software Product Line Cost

In this research, we consider that a product member from a SPL is defined by the set of features F it is made of. This description requires a clear definition of what a feature is. Among the different feature definitions [32], our model is compliant with Batory's definition which states that a feature is an increment of product functionalities [9]. Those features will be used to model the software variability which refers to the ability of a software system or artefact to be efficiently extended, changed, customized or configured for use in a particular context [139].

We influence scope by deciding which features we include. Particularly during the product portfolio scoping phase. Features are a means to realize tasks,

[1] $\mathcal{P}(F)$ is the powerset of a set F which represents a set of features

and thereby assess the cost of a product line needed to satisfy a particular set of tasks. Developers can more easily give an estimation of the development cost of a feature than of the cost to satisfy a goal. On the contrary, they are no good measurements for the willingness-to-pay of customers regarding a product. For reasons mentioned above, we see goals as a better means than features to evaluate revenue that can be generated. When evaluating a software product, customers will not assess features in an isolated way, but how this feature can help to satisfy their requirements, i.e. their goals, softgoals and quality constraints.

For assessing a SPL cost, we have to underline that SPLE consists of two main processes, namely domain engineering (DE) and application engineering (AE). DE is the process of SPLE in which the commonality and the variability of the product line are defined and realized. It results in a Core Assets Base (CAB) which is the basis for the production of products in the product line. AE is the process in which the applications of the product line are built by reusing domain artifacts from the CAB and exploiting the product line variability [117]. In [12], Böckle et al. introduce a cost function for product line based on this distinction.

$$C_{org} + C_{cab} + \sum_{n}^{i=1} C_{unique}(p_i) + \sum_{n}^{i=1} C_{reuse}(p_i) \tag{6.1}$$

where C_{org} is the cost supported by an organization adopting SPLE. C_{cab} is the cost to develop the core assets base suited to support the product line to-be. $C_{unique}(p_i)$ is the cost to develop the unique software part of the product p_i that is not in the core base asset while $C_{reuse}(p_i)$ is the cost to reuse the core assets for the product p_i.

Since the core asset base and the products are described in terms of features, assessing the above cited costs in our framework will require that we evaluate the cost of the features. We assume that C_{org} does not depend on the scope of the considered product line and we therefore do not use it in our optimization. C_{cab} is the cost for developing the set of features which can be reused as part of the product line. Developing a feature as part of the core assets base introduces extra costs such as making it more generic or putting it in a register. C_{reuse} is the cost of reusing a feature integrated in the CAB. Eventually, C_{unique} will be the cost of developping a feature independently of all reuse considerations.

Limited Resources and Discount Rate

The development of a product line is a long term process. Due to limited resources, this development would likely take several years. Taking into account the cost of time and the limited resource is of primary importance for the deter-

mination of the product line scope but more specifically to determine priorities in the development and a release planning.

To take into account the cost of time, we suggest to discount revenues and costs with the Weighted Average Cost of Capital (WACC). This rate calculates an organization's cost of capital in which each category of capital is proportionately weighted. It is often used to discount cash flows and determine the Net Present Value (NPV) of a project.

Regarding the limited resources, we assume that main resources of a SPL provider is its development team. We then consider that the previously described cost function will consume this resource and features will have to be estimated in terms of development effort and not in monetary estimations.

6.4 Optimization Model for the SPL Scope

In this section we describe the mathematical model for finding the optimal scope of a product line. We showed in section 6.3 that the requirement engineering problem of software product line formulated with CORE consisted of three steps. First, determining who are the relevant customers (from possible different domains) and what their needs are. This requires the identification and evaluation of $K_i, T_i \mid\sim G_i, S_i, Q_i$ for different customers. Secondly, defining what the products were constituted of. Answering this question implies that we need to identify and evaluate a set of features F which can be used to derive product members, i.e. $p \in \mathcal{P}(F)$. Thirdly, we need to identify domain assumptions for the product to realize the tasks, i.e. K''. The description of our model follows this distinction. After presenting the objective function, we discuss the constituting features. Then, we deal with goal models to describe customer needs. Finally, we state constraints modeling conditions of tasks realization. The MIP is described in MIP 6.1 and MIP 6.2.

MIP 6.1. First part of the MIP description for SPL Scope.

Constants

Δ	Majoration factor for developing a feature as a generic feature
δ	Reduction factor for reusing a CAB feature in a product
r	Discount rate
c_w	Weekly rate of pay

Sets

P	$= \{1, \ldots, p\}$	Periods
N	$= G \cup T \cup S \cup Q \cup K \cup I$	Concept nodes and inference nodes (I)
F	$= \{1, \ldots, f\}$	Features
F^X	$\subseteq \mathcal{P}(F)$	Sets of features mutually exclusive
F^M	$\subseteq \mathcal{P}(F)$	Sets of features mutually required
F^R	$\subseteq F \times \mathcal{P}(F)$	Sets of required relations between features
J	$= \{1, \ldots, j\}$	Products

Decision Variables

$\sigma_{N,P} \quad = \{\sigma_i \in \mathbb{B} : i \in N \cup I, p \in P\}$
Binary variables representing satisfaction of nodes n during period p.

$\varphi_{F,J,P} \quad = \{\varphi_{f,j,p} \in \mathbb{B} : f \in F, j \in J, p \in P\}$
Binary variables representing the feature f is used in the product j during the period p

$\varphi^{\text{cab}}_{F,P} \quad = \{\varphi^{\text{cab}}_{f,p} \in \mathbb{B} : f \in F, p \in P\}$
Binary variables representing the feature f has been integrated to the CAB
during the period p

$\varphi^{\text{r}}_{F,J,P} \quad = \{\varphi^{\text{r}}_{f,j,p} \in \mathbb{B} : f \in F, j \in J, p \in P\}$
Binary variables representing the feature f was reused from the CAB
for the product j during the period p

$\varphi^{\text{u}}_{F,J,P} \quad = \{\varphi^{\text{u}}_{f,j,p} \in \mathbb{B} : f \in F, j \in J, p \in P\}$
Binary variables representing the feature f was uniquely developed
for the product j during the period p

$w_P \quad = \{w_p \in \mathbb{R} : p \in P\}$
Represents the number of development weeks allocated for period p

Additional Functions to basic MIP 3.1

prod	$: J \to \{n : n \in N\}$	Return the set of nodes of a particular product
c	$: F \to \mathbb{R}$	Time required for developing features

MIP 6.2. Second part of the MIP description for SPL Scope.

Objective function

$$\max \sum_{p \in P} \left(\sum_{n \in N} \frac{\mathsf{u}(n,p)}{r^p} \sigma_{n,p} - \frac{c_w}{r^p} w_p \right)$$

Constraints

Premises constraints
$\forall p \in P, i \in I$:
$$\sigma_{i,p} \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_{x,p}}{|\mathsf{in}(i)|}$$

Conclusion constraints
$\forall p \in P, n \in \{x \in N : |\mathsf{in}_I(x)| > 0\}$:
$$\sigma_{n,p} \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_{i,p}$$

Conflict constraints
$\forall p \in P, c \in C$:
$$\sum_{x \in \mathsf{in}(c)} \sigma_{x,p} \leq 1$$

Mandatory constraints
$\forall p \in P, j \in J, n \in \mathsf{prod}(j)$:
$$\sigma_{n,p} \leq \sum_{i \in \mathsf{prod}(j) \cap M} \frac{\sigma_{i,p}}{|\mathsf{prod}(j) \cap M|}$$

Feature choice
$\forall p \in P, j \in J, f \in F$:
$$\varphi_{f,j,p} \leq \sum_{s \in \{1,\ldots,p\}} (\varphi^{\mathrm{u}}_{f,j,p} + \varphi^{\mathrm{r}}_{f,j,p})$$

Reuse from CAB
$\forall p \in P, j \in J, f \in F$:
$$\varphi^{\mathrm{r}}_{f,j,p} \leq \sum_{s \in \{1,\ldots,p\}} \varphi^{\mathrm{cab}}_{f,p}$$

Budget constraints
$\forall p \in P$:
$$\sum_{f \in F} \mathsf{c}(f) \left(\Delta \varphi^{\mathrm{cab}}_{f,p} + \sum_{j \in J} (\varphi^{\mathrm{u}}_{f,j,p} + \delta \varphi^{\mathrm{r}}_{f,j,p}) \right) = w_p$$

Required features
$\forall p \in P, j \in J, F^r \in F^R$:
$$\varphi_{\pi_1(F^r),j,p} - \frac{\sum_{x \in \pi_2(f^r)} \varphi_{x,j,p}}{|\pi_2(f^r)|} \leq 0$$

Mutually required feat.
$\forall p \in P, j \in J, F^m \in F^M$:
$$\varphi_{z,j,p} - \frac{\sum_{x \in F^m \setminus z} \varphi_{x,j,p}}{|F^m \setminus z|} = 0$$

Mutually exclusive feat.
$\forall p \in P, j \in J, F^x \in F^X$:
$$\sum_{f \in F^x} \varphi_{f,j,p} \leq 1$$

The Product Line

The equation (6.2) is the objective function maximizing the profit which is the difference between the revenue resulting from satisfied goals and the development costs which are evaluated by salary charge. Additionally to MIP 3.1, other constraints are inserted into the model. The first one (equation (6.3)) states that the development is allocated to either domain engineering or application engineering following the cost function previously discussed. Morevoer, it is possible that feature used in a product at a particular period segment would have been reused from the CAB or developed exclusively for this product (development being made at this precise period or in a previous one). This is stated in equation (6.4). However before reusing a feature, it has to be integrated in the CAB (6.5).

$$\max \sum_{p \in P} \left(\sum_{n \in N} \frac{\mathrm{u}(n,p)}{r^p} \sigma_{n,p} - \frac{c_w}{r^p} w_p \right) \tag{6.2}$$

subject to:

$$\forall p \in P : \sum_{f \in F} \mathrm{c}(f) \left(\Delta \varphi^{\mathrm{cab}}_{f,p} + \sum_{j \in J} (\varphi^{\mathrm{u}}_{f,j,p} + \delta \varphi^{\mathrm{r}}_{f,j,p}) \right) = w_p \tag{6.3}$$

$$\forall j \in J, f \in F, p \in P : \varphi_{f,j,p} \leq \sum_{s \in \{1,\ldots,p\}} (\varphi^{\mathrm{u}}_{f,j,p} + \varphi^{\mathrm{r}}_{f,j,p}) \tag{6.4}$$

$$\forall j \in J, \forall f \in F, \forall p \in P : \varphi^{\mathrm{r}}_{f,j,p} \leq \sum_{s \in \{1,\ldots,p\}} \varphi^{\mathrm{cab}}_{f,p} \tag{6.5}$$

Additional considerations about features need to be stated. Feature dependencies and interactions are important constraints when developing a product line. We can identify some patterns when considering feature dependencies. On the one hand, we can identify common features which are features belonging to all product members of the SPL. On the other hand, as highlighted in [96] variable features fall into three categories: alternative, OR and optional features. Moreover, there are some dependencies among features. These can take several forms such as “mutually requires”,“requires” or “mutually excludes”. Due to some operational dependencies [96], some of those dependencies and interactions have to be identified since they will have significant implications in the development of the system. Here we present the instantiation of those feature interactions into linear constraints.

The “mutually excludes” constraint expresses that some features can not be active at the same time. E.g., consider a security system. Because of some interferences between waves, radar motion detectors can not work at the same time as

wireless networks. In terms of linear constraints, (6.6) expresses that for all sets of features that are mutually exclusive $F^X \subseteq \mathcal{P}(F)$:

$$\forall p \in P, \forall j \in J, \forall F^x \in F^X : \sum_{f \in F^x} \varphi_{f,j,p} \leq 1 \tag{6.6}$$

We can also model the "mutually requires" constraint between sets of features, i.e. the feature can not be used independently. Let's consider all sets of mutually required features $F^M \subseteq \mathcal{P}(F)$, the linear constraint is:

$$\forall p \in P, \forall j \in J, \forall F^m \in F^M : \varphi_{z,j,p} - \frac{\sum_{x \in F^m \setminus z} \varphi_{x,j,p}}{|F^m \setminus z|} = 0 \tag{6.7}$$

with $z \in F^m$.

For example, consider that $F^m = \{\mathsf{F1}, \mathsf{F2}, \mathsf{F3}\}$ which states that those three features cannot be used independently, then the constraint will be:

$$\varphi_{\mathsf{F1},j,p} - \frac{\varphi_{\mathsf{F2},j,p} + \varphi_{\mathsf{F3},j,p}}{2} = 0$$

Finally, we can also model the "requires" constraint. It is necessary if a feature F_i requires a set of other features to be part of the product while the latter can be integrated in the product without F_i. We can model this constraint as following: Let's consider the set of tuple $F^R \subseteq F \times \mathcal{P}(F)$ where the first member of the tuple is the feature requiring the second member of the tuple. For example, $(\mathsf{F1}, \{\mathsf{F4}, \mathsf{F5}\})$ means that a product consisting of F1 should also integrate F4 and F5.

$$\forall p \in P, \forall j \in J, \forall F^r \in F^R : \varphi_{\pi_1(F^r),j,p} - \frac{\sum_{x \in \pi_2(F^r)} \varphi_{x,j,p}}{|\pi_2(F^r)|} \leq 0 \tag{6.8}$$

with $\pi_i(X)$ being the i^{th} projection map of tuple X.

Goal Models

Since we are scoping our product portfolio, mandatory constraints need to be handled with care. Indeed, in a scoping context, a mandatory constraint does no more mean that the goal must be satisfied in the final software product line. Instead, it means that *if* we decide to provide a product for this requirement problem (i.e. customer), then the product will be legal only if it satisfies those mandatory goals. This is formulated as following: let's consider the function prod which returns all concept nodes of the goal model related to a specific product. Then, the mandatory constraints becomes:

$$\forall p \in P, j \in J, n \in \mathsf{prod}(j) : \sigma_{n,p} \leq \sum_{i \in \mathsf{prod}(j) \cap M} \frac{\sigma_{i,p}}{|\mathsf{prod}(j) \cap M|} \tag{6.9}$$

This equation means that a node cannot be satisfied if all mandatory nodes have not been satisfied. We take here the assumption that the first release of the product needs all its mandatory nodes (in that we are compliant with the definition of the *must-be* in the Kano model [90]).

Tasks realization

Modeling task realizations follows the same pattern as goal refinement. For example to model the following task realization $(F1 \wedge F2) \vee (F1 \wedge F3) \rightarrow T1$ which states that the task T1 will be realized either with the features F1 and F2 or with F1 and F3 will use the following equations:

$$T1 \leq I1 + I2 \tag{6.10}$$

$$I1 \leq (F1 + F2)/2 \tag{6.11}$$

$$I2 \leq (F1 + F3)/2 \tag{6.12}$$

where I1 and I2 are inference nodes. These additional task realization constraints have been added in the *premises* and *conclusion* constraints.

6.5 Further Commonality and Variability Analysis

The output of the MIP consists among others of different sets of features satisfying each market segment. The model determines if those features are developed during domain or application engineering.

Actually, the software products satisfying the different market segments are not the members of the product line for two reasons. First, they can include features developed exclusively for these market segment. Consequently, the product has more features than possible product line members. Secondly, it is not mandatory to base the feature constraints definition of the product line on the strict interpretation of the model output. For example, assume that the model determines the four following products: $p_1 = \{F1, F2, F3, F6\}$, $p_2 = \{F1, F2, F4, F6\}$, $p_3 = \{F1, F2, F5\}$ and $p_4 = \{F1, F2, F4, F7\}$. Moreover, it considers that $\{F1, F2, F4, F6\} \subseteq$ CAB. A strict definition of the software product line described in a feature diagrams could be one such as in Fig. 6.1(a). However, SPL engineers could decide to eventually design a software product line such as in Fig. 6.1(b) which also allows the definition of $p_x = \{F1, F4, F6\}$

The different decisions taken about the feature constraints in this commonality and variability analysis (as well as possible feature aggregation) will result in the definition of the Product Line variability which describes the variation between

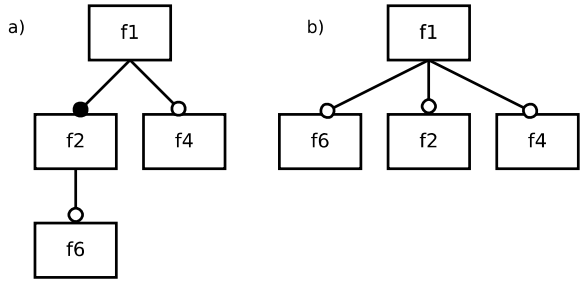


Figure 6.1: Example of various feature diagram definitions

the systems that belong to a SPL in terms of properties and qualities. This analysis is required to disambiguate the two types of variability [105].

This discussion underlines that although we assume some structural and operational constraints identified during pre-optimization steps, a further commonality and variability analysis of the core assets base is required after processing the optimization. Acher et al. suggested a method able to extract feature diagrams from products description [4].

6.6 Context-Aware Considerations

In this section, we describe how to adapt the MIP to various SPLE situations. First, we make a distinction between different market strategies. Then, we show how to take into account clear separation between domain and application engineering teams. Finally, we deal with SPL development from legacy systems.

Mass-Customization or Mass-Markets

As highlighted by K. Schmid in [129], we can identify two product strategies: the *mass-customization* and the *mass-markets*. Depending on the strategy, constraints of the mathematical model will be different.

In a mass-market situation, the product is mainly defined by the producing organization in order to sell the product in a market segment to a large number of customers during a certain span of time. Consequently, if the goal value were an annual expected revenue, revenues from the model can be added each year. Moreover, this product can be improved (with new features) and a new release can be sold to this segment. This is typical in mobile applications or in *Software as a Service* (SaaS) solutions. This product strategy is the default situation in our

mathematical model. Each goal model represents the aggregated needs of a particular market segment and goal value are predictions of yearly sells.

In a mass-customization situation, each product is designed for the specific needs of a customer. Consequently, a product is sold only once and estimated revenue can be obtained only once. It means that each goal model captures the needs of a particular customer, which implies that it can be satisfied only once. However, we can distinguish two situations. In the first, once a product is derived and deployed, customer needs are considered as satisfied and no more change will be accepted. This situation is common in the waterfall methodology and is known as a *big bang* release [125]. We will then add these sets of constraints:

$$\forall n \in N : \quad \sum_{p \in P} \sigma_{n,p} \leq 1 \tag{6.13}$$

$$\forall j \in J, n \in \mathsf{prod}(j) : \quad \sum_{p \in P} \sigma_{n,p} \leq \gamma_{j,p} \tag{6.14}$$

$$\forall j \in J : \quad \sum_{p \in P} \gamma_{j,p} \leq 1 \tag{6.15}$$

where $\gamma_{j,p}$ is equal to 1 if the product j is delivered at period p and (6.13) states that a goal can be satisfied only once through the whole set of considered periods and (6.14) and (6.15) state that there is only one release of the product dedicated to the customer j.

In the second situation, a product can be updated and a new release can be deployed. In this case, we can consider that unsatisfied goals can still be satisfied at a later point in time (i.e. in a future release). In this case we have to add only the set of equations (6.13). This situation occurs in iterative methodologies.

Separation of Domain and Application Engineering Teams

Organizations can for various reasons [101] decide to separate or merge domain and application engineering teams. Those distinct situations can be integrated in our mathematical model. Merged teams is the default situation where w_t in (6.3) represents the available development resources for one period dedicated to both domain and application engineering.

In order to make a distinction between domain engineering and application engineering, we have to replace the decision variables w_t by w_t^D and w_t^A which represent respectively development weeks for domain engineering and applica-

tion engineering. Then, (6.3) has to be replaced by:

$$\forall p \in P : \qquad \Delta \sum_{f \in F} \mathsf{c}(f) \varphi^{\mathrm{cab}}_{f,p} \leq w^{D}_{p} \tag{6.16}$$

$$\forall p \in P : \qquad \sum_{f \in F} \left(\mathsf{c}(f) \sum_{j \in J} (\varphi^{\mathrm{u}}_{f,j,p} + \delta \varphi^{\mathrm{r}}_{f,j,p}) \right) \leq w^{A}_{p} \tag{6.17}$$

Further situations can also be modeled, e.g. two basis team dedicated to domain and application engineering helped with a flexible team which could be work on both domain and application engineering. It is interesting to notice that this paragraph deals with an additional RI, namely RI6 *"How to distribute the responsibility for the satisfaction of requirements to the system-to-be, systems it might interact with, and people in its environment?"*. However, this problem of distribution of responsibilities will be deeper investigated (in another context) in Chapter 8.

Encapsulation of Legacy Systems

Developing a software product line on the basis of legacy systems by wrapping some components is a common situation in SPLE. For instance, as described in [101], a legacy component can be first wrapped and integrated to a product line before it would later be replaced by a new component.

This situation is modeled by adjusting the cost of feature integration to the CAB. For example, consider a financial calculation module which was present in a legacy system. We can then consider two features realizing the same tasks where f_1 is the legacy module and f_2 is an similar module developed from scratch. Then we have that the cost to integrate f_1 to the CAB will be lower than the cost to integrate f_2. We also have to fix the variable $f^{\mathrm{u}}_{1,m,t} = 0$ since the legacy module cannot be developed uniquely for a single product.

6.7 Change and Iterative Development

We advocate to apply this model in an iterative and incremental development process and not in a predictive way. Since all products will not be developed at once and that the CAB will incrementally grow as new products are developed, it would be useful to frequently reevaluate the scope.

It results from two facts. First, developing a software product line is a long term process and the product line life cycle is often largely longer than for a simple software product. So, all products can not be developed at the same time. Secondly, it is impossible to exactly predict and evaluate customer needs several

years in advance. Environment changes are frequent and they can affect previously stated customer needs. Additionally the probability that changes occur is a function of the considered time span. Considering a too long time will obviously result in unexpected change.

Our suggested method can not be considered as an exact predictive method to be applied in the early phase of a product line life cycle without any later changes. It has to be used in an iterative development process and results have to be considered as a sketch for further investigation. Those consideration justify our use of goal models since they allow to model high-level requirements (which are less likely to change). Instantiated with an incremental and iterative development method, our model can help to control the development and release planning since we can integrate environment changes in goal models and we can model the development progress of the CAB. E.g. a feature already developed and integrated to the CAB can get a zero development cost and a positive reuse cost.

6.8 Application on MystShop Case

In this section, we apply our method to the MystShop case. In the first part of this section, we provide a software product line context to the case and we model it with goal model. In a second part, we present the result of the software product line scope optimization (performed by AnalyticGraph).

Problem Statement

The platform developed for MystShop could be used for many other domains. For instance, a research who performs a survey could use similar features in order to design the survey and ask many people to answer it. If not all features can be reused, we can point among other the *questionnaire designer* feature (able to design the checklists) as well as a *csv export* feature. As an illustration, we will consider three additional markets for the MystShop platform: a research survey platform, a poll widgets designer and a meeting scheduler platform.

The poll widget designer consists of a platform which will be used to configure a poll consisting of one question and a set of possible answers. This poll will be displayed as a widget in websites in order to survey the users. Such widgets are frequent on news websites.

The meeting scheduler platform would be aimed at getting availabilities from a list of participants in order to set an event date. An example of such platform is `www.doodle.com`.

Simple goal models for each potential product are depicted in Fig. 6.2. The MystShop goal model is in the top-left corner. The poll widget designer and the

research survey designer are respectively in the top-right and bottom-right corner of the figure. The meeting scheduler plateform goal model is in the bottom left corner.

On the basis of these goal models, we identified a set of 13 coarse-grained features able to realize tasks identified in each goal models. They are described in Fig.6.3. The CAB integration cost and the reuse cost are evaluated by applying respectively a design-for-reuse factor and a design-with-reuse factor on the traditional development cost. We assumed for this example those factors to be respectively $\Delta = 1.5$ and $\delta = 0.2$. Some feature dependencies were identified before the optimization of the product portfolio.

In this problem, we consider two periods of concern and a limited budget of 300 units per period.

Results presentation

The entire model was modeled and resolved with AnalyticGraph.com. Results are depicted in Tab.6.1, Fig.6.5 and Fig.6.4. The optimal value is 293. It corresponds to the cumulative revenue acquired from the satisfaction of goals minus the cost of implementing the underlying features.

The first period should be dedicated to create a CAB consisting of F182, F186 and F244. Each of these features are required at least in three distinct products. The rest of the available work is allocated in configuring and developing features for the mystery shopping product. Some are directly reused from the CAB (F182, F186 and F244) and others are developed specifically for this product (F187 and F194). This product is selected because no additional features should be integrated into the CAB in order to create it.

During the second period, the CAB is extended with two additional features (F190 and F193). The survey product as well as the poll product require the development of specific features while the meeting scheduler is only based on features from the CAB. It does not mean that no effort are required for this latter product. Some work have to be considered to customize those CAB features. It is also interesting to notice that an additional feature is integrated in the mystery shopping product. It is F188, the "File uploader" feature. It allows to provide customers with the additional service of pictures of mystery visits.

We see in Fig.6.4 that break even is reached after the second period. The expected operating profit over the two periods is 293.

Fig.6.5 shows a possible feature diagram for describing the product line variability after an analysis of the variability and commonality between products.

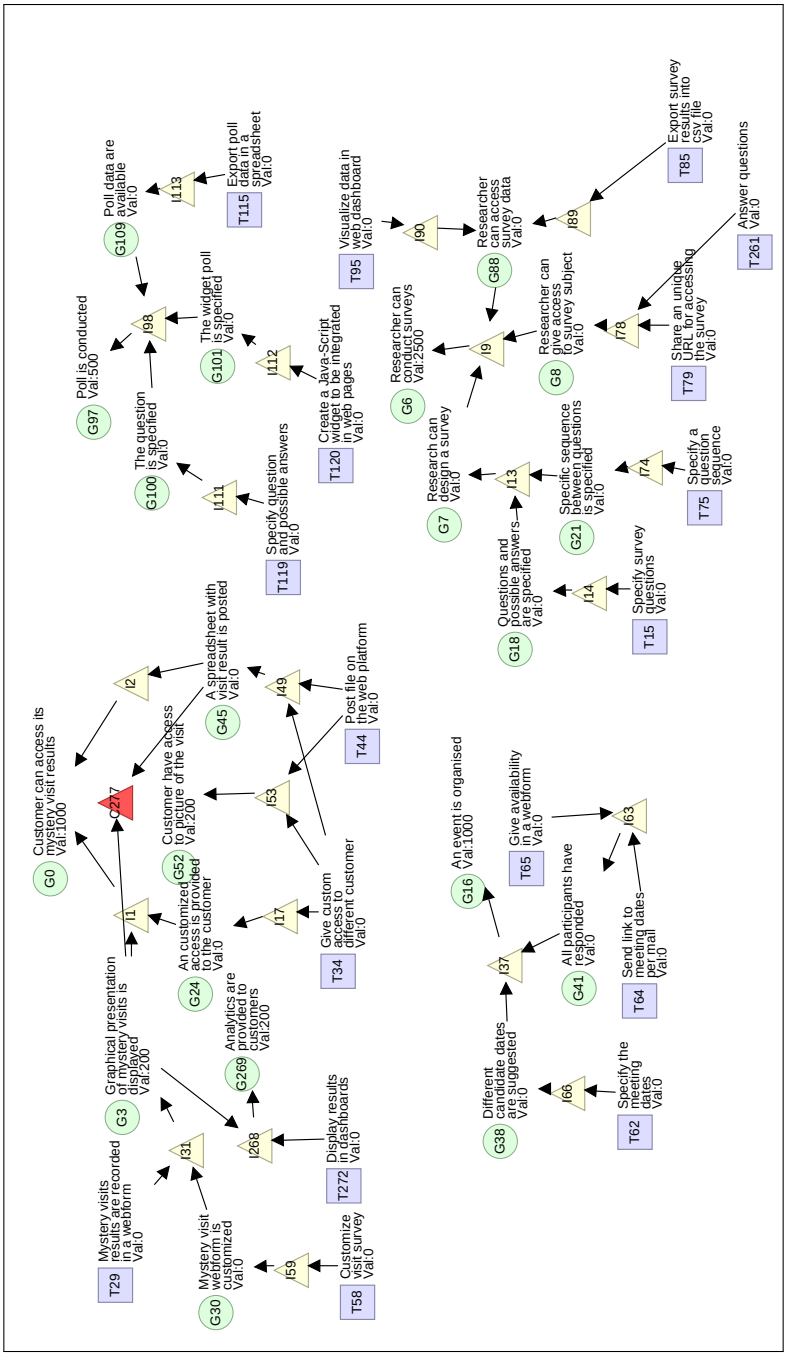


Figure 6.2: Goal Model of MystShop Requirement Problem. Accessible at `http://analyticgraph.com/dev/?g=T1oQ2C1ZPC`

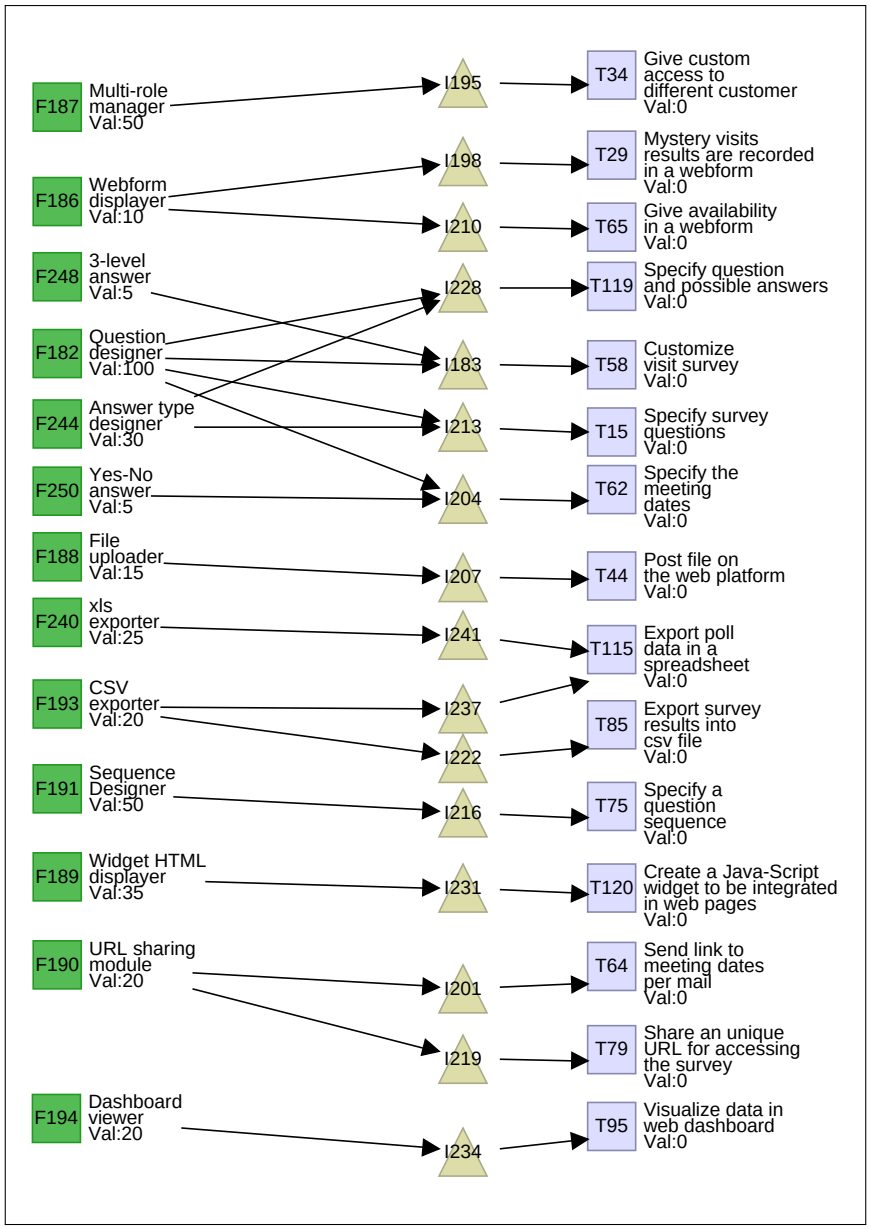


Figure 6.3: List of identified features and task realization links.

Table 6.1: The product portfolio planning - Increments between periods are bolded, r = reuse, u = uniquely developed

P.	Segment	Satisfied goals	Product composition										CAB
			F187	F186	F248	F182	F244	F250	F188	F240	F193	F191	F189	F190	F194	
1	MystShop	{G0, G3, G24, G30, G269}	**u**	**r**		**r**	**r**								**u**	{F182, F186, F244}
	Surveys	∅														
	Poll widget	∅														
	Meet. sched.	∅														
2	MystShop	{G52}	**u**	**r**		**r**	**r**	**u**							**u**	{F182, F186, F244, F190, F193}
	Surveys	{G6, G7, G8, G18, G21, G88}		**r**		**r**	**r**				**r**	**u**		**r**		
	Poll widget	{G97, G100, G101, G109, G153}				**r**					**r**		**u**	**r**		
	Meet. sched.	{G16, G38, G41, G109, G153}		**r**		**r**	**r**							**r**		

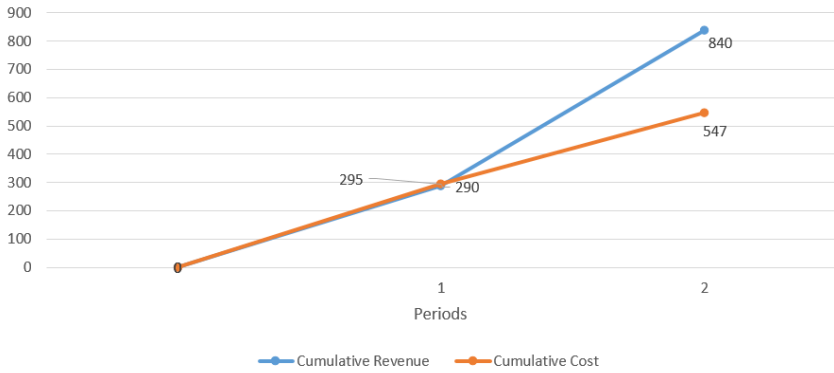


Figure 6.4: Expected discounted cumulated cash flows

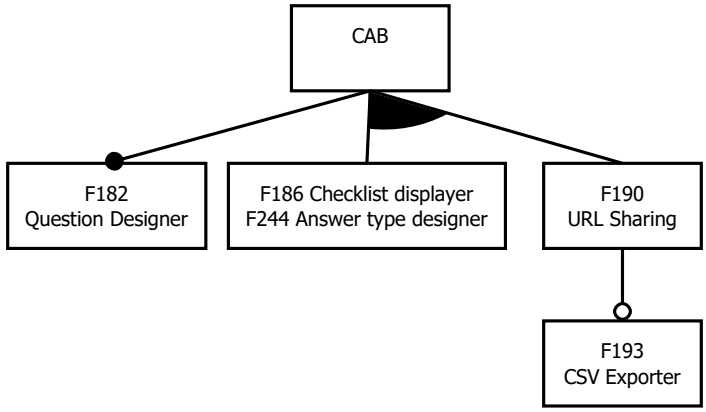


Figure 6.5: A possible feature diagram for the MystShop Product Line

6.9 Related Work

Although several product line scoping methods have been designed, few address the optimization goal of the product portfolio scoping on the basis of the profitability concern.

Recently, J. Müller suggested a mathematical program able to optimize the product portfolio on the basis of a profit maximization objective [106]. His model

is based on previous work in product lines (i.e. not software) and on conjoint analysis. Although the idea to use a mathematical program to optimize the SPL scope is similar to ours, the underlying assumptions of the two models are different. First, he assumes that the features constitution of each product have already been established before optimizing the scope. His model only optimizes the selection of the product-segment pair given the already constituted product family and a set of relevant segments. Moreover, it does not introduce any time considerations what prevents any possibility of setting development priorities or release planning which is however a primary concern in SPLE.

Several surveys of Software Product Line Scoping have been performed. In their survey [79], I. John and M. Eisenbarth identify 16 scoping approaches but they highlight that scoping optimization has only been partially addressed. Moreover none of them integrate the optimization of the three types of scoping. They also highlighted that among the five methods they identified as addressing the product portfolio analysis [27, 74, 117, 140, 142] only one [27] also addresses asset scoping (as we do with our model).

The Quality Function Deployment Product Portfolio Planning suggested by Helferich [74] elicits required features of products of a SPL and asks engineers about technical feasibility. It also allows to identify customer segments. However it does not consider revenue and cost aspects and therefore cannot be used for profit maximization. However, this method can usefully be viewed as complementary to ours since it identifies features and can be used to match them with goal models.

In [117], Niehaus et al. present a Kano model which allows to design customer-oriented SPL. It is focused on the product portfolio planning and consequently do not consider any asset scoping.

The issue of a release planning is addressed in [140, 142] but they do not address neither composition of product portfolio or asset scoping.

Moreover as highlighted by J. Müller none of those five methods integrate market and cost perspectives which prevents any profitability consideration.

6.10 Conclusion and Further Work

In this chapter, we suggested that instances of **RI14** could relevantly be modeled as ROP in CORE. More precisely, we focused on the problem of portfolio scoping of a Software Product Line. This contributes to **RQ1**. Indeed, our scoping model is based on the description of customer needs in terms of goals to satisfy and tasks to realize. This use is justified because first, we showed that goals are better than features to capture the customer willingness-to-pay and secondly, they are able to capture the alternative solutions of a requirement problem what is useful to find

large commonalities between market segments. Features are then identified and linked to tasks they support during the refinement process.

Current methods to deal with SPL suggest to describe it in terms of features. In order to link both feature diagrams and goal models, we introduced task realization links which indicates that some features are required to execute some tasks. We extend our basic model with feature nodes, product labels and requires/excludes relationships. This contribution answers **RQ2**.

Finally, we proposed that when scoping, identification and evaluation of the three types of scope (i.e. product portfolio, domain and assets scoping) can be performed separately but the optimization of those scopes has to be performed in an integrated model. Consequently, we proposed a mathematical program able to optimize the software product line scope and sketch both a development and a release planning. Our method is based on the assumption that all software product line objectives are eventually driven by the same general purpose: profit maximization. Revenues are function of the customer satisfied needs and costs are function of the feature development effort. This is the contribution to **RQ3**.

We showed that our mathematical model can be instantiated in several contexts such as a market customization strategy or a mass-customization strategy. It can deal with Software Product Line development from scratch as well as from the basis of legacy software. The output of our model are a good basis for further commonality and variability analysis. We also illustrated its applicability with the MystShop case by extenting AnalyticGraph.com to support all previous contributions.

In further work, we should consider the following extensions. We limited our definition of features to Batory's definition which allows to simplify the relation between feature and cost. However, a more complete feature definition would require to take into account the non-monotonicity between features and SPL cost function. We need to integrate risk management in our model with for example stochastic models.

7 Optimization with Cost Estimation Method

7.1 Forewords

This chapter suggests a third optimization problem which focuses on **RI7** *"How to estimate costs, risks, and deadlines for making systems that satisfy requirements"*. It is aimed at introducing an existing cost estimation method into the optimization model. Indeed, though we already discussed cost in previous chapters (indirectly in Chapter 5 through Story Points as well as in Chapter 6 where we introduced the Böckle's cost function for SPL), we did not used a formal cost estimation method. The purpose of this Chapter is then to show that it is possible to use our framework in combination with existing formal estimation methods such as COCOMO (and COCOMO II) [19], COSYSMO [143], Function Point Analysis [6] or any other model-based estimation method.

This Chapter is an extension of the previous ROP dealing with Software Product Line. It is aimed at being published in a journal.

7.2 Introduction

Context: Using Formal Cost Estimations as Inputs for a Cost Minimization

As discussed in the previous chapter, one of the main advantages of deploying Software Product Lines is to reduce the total cost. The most widely accepted SPL cost function was suggested by Böckle et al. [12] and is equal to:

$$C_{org} + C_{cab} + \sum_{n}^{i=1} C_{unique}(p_i) + \sum_{n}^{i=1} C_{reuse}(p_i) \tag{7.1}$$

As it is, this cost function cannot be practically used to estimate SPL cost. Indeed, no guidelines or methodology are provided on how to estimate C_{org}, C_{cab}, C_{unique} or C_{reuse}. In this research, we use this function as a starting point and apply a more practical cost estimation method. The idea is to map our approach with an

existing cost estimation method and being able to apply the guidelines suggested by the methodology to, we hope, more accurately estimate the software cost.

Here, the research purpose is not to choose the most effective method but rather to show that an existing estimation method can be used in collaboration with our model. Then, it does not focus on the selection of the method but rather tries to show how one of them can be used and how it can be integrated it in our framework.

The literature on cost estimation methods identifies 2 main famillies of effort estimation methodologies, namely, model-based or expert-judgment-based estimations [82]. What distinguish them is mainly the quantification step, i.e. the final step that transform the input into the effort estimate.

In expert-judgment, effort estimations are asked to experts (project managers, advisors, senior developers...). There exists different mechanisms to confront diverging judgements such as the Delphi method [102, 17] or the Poker Planning [104]. By suggesting to use of Poker Planning in Chapter 5, we already showed how results of such mechanisms could be integrated in our framework (that is, by labeling each node with the result of the estimation process). In this Chapter, we are more interested to show how a formal method could be used.

Although one could say that model-based are rarely used in practice [81] and research is currently switching its attention from formal-based to judgment-based estimations [83], researchers still admit that if carefully used model-based estimations have still interests [82, 93]. One of them is that with formal estimation methods parts of the counting process can be automatized. For instance, if a Use Case analysis or an Entity-Relationship (ER) analysis have been performed during the requirement phase, the results of those analysis can be used as inputs of a model-based cost estimation method in order to semi-automatically estimates the cost.

Scope: Function Points Analysis as Cost Estimations

Beyond those two families, there is no widely accepted categorisation of estimation methods although they are numerous. Through surveys, some authors have suggested different categorisations. In a survey, Boehm et al. distinguished six main types of estimation techniques, respectively: Model-based, Expertise-Based, Learning-Oriented, Dynamics-Based, Regression-Based and Composite [15]. In their systematic literature review [83], Jorgensen and Shepperd classified the cost estimations as follows: Regression, Analogy, Expert Judgment, Work breakdown, Function Point, CART, Simulation, Neural Network, Theory, Bayesian, Combination of estimates and Others. In both cases, these categories still hide countless methodologies.

In this Chapter we selected Function Point Analysis as the estimation method. Other formal estimation methods could have been selected such as line of code counting (such as COCOMO) or Use Case Points (UCP). The former was simply discarded because of the difficulty to estimate line-of-code. The latter could have been selected but we decided to apply Function Point Analysis (FPA) because UCP required actors to be identified what was not done in this particular chapter. Nonetheless, UCP could be a more interesting candidate for ROP integrating actor concepts (such as Chapter 8). Eventually, using Function Points to estimate effort of Software Product Line has already been suggested although not really investigated [118].

Among the FPA, we can distinguish several techniques. Five of them, whose evolution is depicted in Figure 7.1, have received ISO standardisations:

- IFPUG (ISO/IEC 20926:2009),
- Mark-II (ISO/IEC 20968:2002),
- NESMA (ISO/IEC 24570:2005),
- COSMIC (ISO/IEC 19761:2011),
- FISMA (ISO/IEC 29881:2010).

Aside from these standards, there also exist numerous other methods. In this Chapter, we selected the implementation provided by the International Function Point Users Group (IFPUG) [2]. Two arguments justified this choice. First, the IFPUG method is among the most popular implementations of FPA [122, 109]. Then, it is the method that was selected by the OMG[1] as basis for their *Automated Function Points* specification which provides a standard for automating the Function Point counting [31]. In order to ease the distinction between the general of Function Point Analysis and the specific implementation by IFPUG, we will hereafter call the latter Function Point Counting (FPC).

The IFPUG's FPC is described as a *"method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user"* [2]. Like any other FPA techniques, FPC presents the characteristics to be unrelated to the languages and/or tools used to develop the software project. FPC is designed to measure business-type applications; it is not appropriate for other types of applications such as technical or scientific applications. Moreover, it has few capabilities to estimate real-time applications (see the COSMIC methodology for more information about this point). However, FPC function points can

[1] Object Management Group - `www.omg.org`

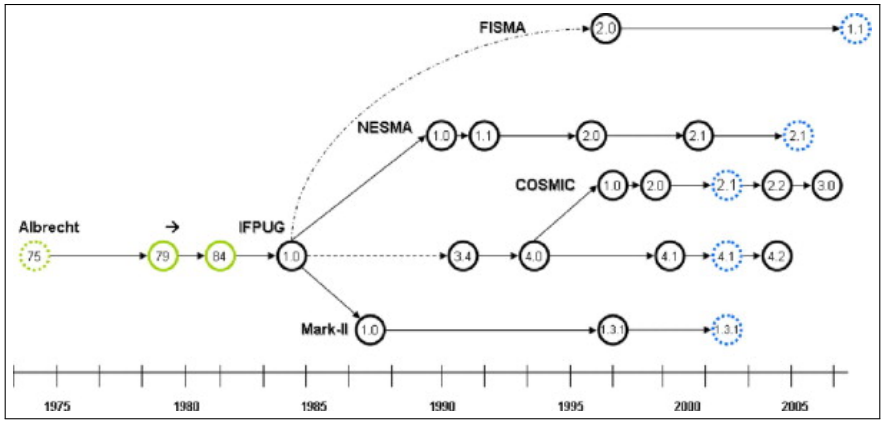


Figure 7.1: Evolution of Function Point Analysis. From Cuadrado-Gallego et al.[39]

be estimated from requirements specifications or design specifications, thus making it possible to estimate development effort in the early phases of development [103]. A comprehensive methodology on how to apply the FPC methodology is described in the IFPUG manual [2].

Although choosing FPC was quite arbitrary, it is interesting to notice that there is a rich literature providing conversion methods from FPC to different estimation method: to COSMIC Points [39, 55, 51], line of codes [80], Use Case Points [38], etc.

Finally, before going further in the description of the problem, we would like to inform the reader that despite SPLE massively uses the concept of feature, Feature Points should not be considered in this context. As discussed in [5], it is not widely accepted and moreover it has a specific focus on algorithm intensive application (which is not the case of systems under consideration in this chapter). Features from SPLE and Feature Points actually do not cover the same meaning.

Problem: Determining Product Portfolio Minimizing the Cost

Applying FPC as a cost estimation method in the context of minimizing the cost of a SPL Portfolio rises multiple difficulties. First, concepts used in both frameworks are different. SPL are described in terms of features which belongs to specific products or to the CAB, while the FPC is based on the counting of Data Functions and Transactionnal Functions. Moreover, FPC is focused on the cost estimation of a single one application while SPL deals with a portfolio of applications.

Nonetheless, using FPC for SPLE (which will be decribed in details later in this chapter) is still valid since it has been showed that features could be mapped to variation points [105] and variation points can be used to reference functional description of systems [117]. The latter being the input of FPC.

Those difficulties have to be added to the main problem which is determining the portfolio which minimizes the cost estimated with FPA.

Contribution: A MIP to Minimize Cost Estimation

In this chapter, we first suggest an approach to model a SPL portfolio and its candidate features the cost of which is estimated with FPA. This first contribution shows how to map goals, features and FPA primitives. Then, we provide an optimization model able to determine the portfolio composition which minimizes the cost.

The SPL portfolio modeling is based on a combined use of (1) goal model to describe the problem space, (2) feature diagrams to describe the solution space and (3) entity-relationship diagrams to decompose features into FPA primitives.

The optimization model is a MIP which minimizes the SPL total cost by deciding which features should integrate the CAB based on their cost estimations.

Limitations: Increasing the Need of Information

The approach described in this chapter particularly differs from other optimization models discussed in this thesis. The main difference results from the amount of inputs required to operationalize the model. Indeed, while models presented in Chapters 3, 5 and 8 can be used in a lightweight style, the currently suggested model requires more information to be provided and then requires more in depth RE.

Consequently, if the former could be considered in short iteration development, the latter will require longer analysis. It means that the time spent at the early phase will be higher and deliverable will be delayed.

This is explained because, first, software product lines requires in essence more prediction and systematic planning [113]. Secondly, using formal cost estimation method also requires more inputs [82].

In consequence, the suggested model can be valuable only if it has been automatized as much as possible. So, the reader needs to keep in mind that, aside from the apparent complexity of the current model, many computation steps described hereafter would be automatically performed.

7.3 Software Product Line Cost

Similarly to the previous chapter, we consider that a product member from a SPL is defined by the set of features F it is made of. This description requires a clear definition of what a feature is. Among the different feature definitions [32], our model still uses Batory's definition which states that a feature is an increment of product functionalities [9]. Those features will be used to model the *software variability* which refers to the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context [139].

We influence the scope by deciding which features we include. They are considered as a means to realize tasks, and thereby influence the cost of a SPL needed to satisfy a particular set of tasks. Developers can more easily give an estimation of the development cost of a feature (e.g. with function points) than of the cost to satisfy a goal since the latter can have alternative solutions.

In FPA, measurements are based on the counting of Data Functions (DFs) and Transactional Functions (TFs) :

Data Functions: Data functions represent the functionality provided to the user to meet internal and external data requirements.

Transactional Functions: Transactional functions represent the functionality provided to the user to process data.

When using both SPL principles with FPA, it is important to underline that SPLE consists of two main processes, namely domain engineering and application engineering. The former is the process of SPLE in which the commonality and the variability of the SPL are defined and realized while the latter is the process in which the applications of the SPL are built by reusing domain artifacts and exploiting the SPL variability [117]. Böckle et al. introduce a cost function for SPLs based on this distinction [12].

$$C_{org} + C_{cab} + \sum_{n}^{i=1} C_{unique}(p_i) + \sum_{n}^{i=1} C_{reuse}(p_i) \tag{7.2}$$

where C_{org} is the cost to an organization for adopting SPLE. C_{cab} is the cost to develop a core asset base (CAB) suited to support the planned SPL. $C_{unique}(p_i)$ is the cost to develop the unique software part of the product p_i that is not in the core asset base while $C_{reuse}(p_i)$ is the cost to reuse the core assets for the product p_i.

Since the Core Asset Base and the SPL products are described in terms of features while FPA uses Data Functions (DF) and Transactional Functions (TF), it is

important to relate both frameworks. We suggest that a feature (i.e. a product increment) should be described as a set of both DF and TF. We assume that Data Functions can be shared by different features while Transactional Functions are partitioned into features (i.e. a TF cannot be shared by several features). As we discussed in the previous chapter, it is possible to describe Böckle's cost functions in terms of features which will be later decomposed into DF and TF. It is then important to describe how to apply the counting procedure of FPA in the context of features.

According to the Function Point Counting Manual [2], FPC is *accomplished using the information in a language that is common to both user(s) and developers*. It implies that both point of view have to be taken into consideration when counting function points. The user view has to be established during the *Initial User Requirement* phase. We describe a user view as a description of the user's business needs in the user's language. Since, the FPC manual does not prescribe a particular user language, we use goal models for this. Then developers will translate the user information into technical-related information in order to provide a solution (the technical-related information we suggest to use is the ER language which relates to relational DBMS [28]). This phase is referred as the *Initial Technical Requirements*. Those both views form the *Final Functional Requirements* which can be used as input for the FPC which consists in mapping those two perspectives.

The first phase of our scoping method is the definition of goal models. Those goal models are user views of business goals. It corresponds to Initial User Requirements in the FP framework. On this basis, developers analyze several possible solutions which result in the determination of features composed of entities, relationships and transactional functions on those entities. Features are then mapped to goal models to ensure goal satisfaction. Since they result from the developer's analysis of goal models, features are a common view between users and developers which makes them ideal candidates to be the basis for the FPC in the early stage of SPL scoping. So, for each identified feature, engineers will have to identify data functions and transactional functions.

For instance, starting from the MystShop example depicted in Fig. 6.2, let's consider the feature F186, entitled *Webform displayer* which allows the user to fill the checklist online. This feature has been identified by developers to potentially take part in the satisfaction of different goals. There are several Data and Transactional Functions related to this feature. For example, we can identify logical files related to *questions* and *answers*. A possible Transactional Function would be *Save answers* which would consist in saving the value of the different questions in the database. Nonetheless, using Function Point Counting on the basis of features to estimate cost during SPL scoping presents two main difficulties. The first one will be referred as the *FPC for scoping problem* while the second one will be referred as

the *FPC for SPLE problem*.

Function points analysis assumes that we have defined the application boundary before starting counting. However, scoping is by definition the process which determines this boundary. Before the resolution of the scoping model, software products have not well defined boundaries (i.e. we do not know at this moment which features are in or out from the product definition). Consequently, this model will have to take into account the different estimations resulting from integrating some logical file as internal or external of the product. This is the *FPC for scoping problem*.

Moreover, as we deal with Software Product Line and not traditional software development, we will also have to take into account that some features of the SPL are developed from the same CAB. Consequently, we have to consider that if there are n products in the SPL, the estimated total cost of the whole SPL will not be the sum of the function points of the n products. On the contrary, we hope that it will be lesser since some logical files and transactional functions will be related to the CAB and products will only interface with functions already developed in the CAB. This is the *FPC for SPLE problem*.

Data Functions

There are two types of data functions to identify: Internal Logical Files and External Interface Files:

Internal Logical File: An Internal Logical File (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted.

External Interface File: An External Interface File (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application.

Each logical file has a number of Data Element Typess (DETs) which are associated to it. DETs are unique user recognizable, non -repeated field. Although, we speak about files, it does not mean that these files are actual physical files. It refers more to logically related groups of data.

The distinction between ILF and EIF is based on the application boundary. However, due to the *FP for scoping issue*, this distinction become more complex

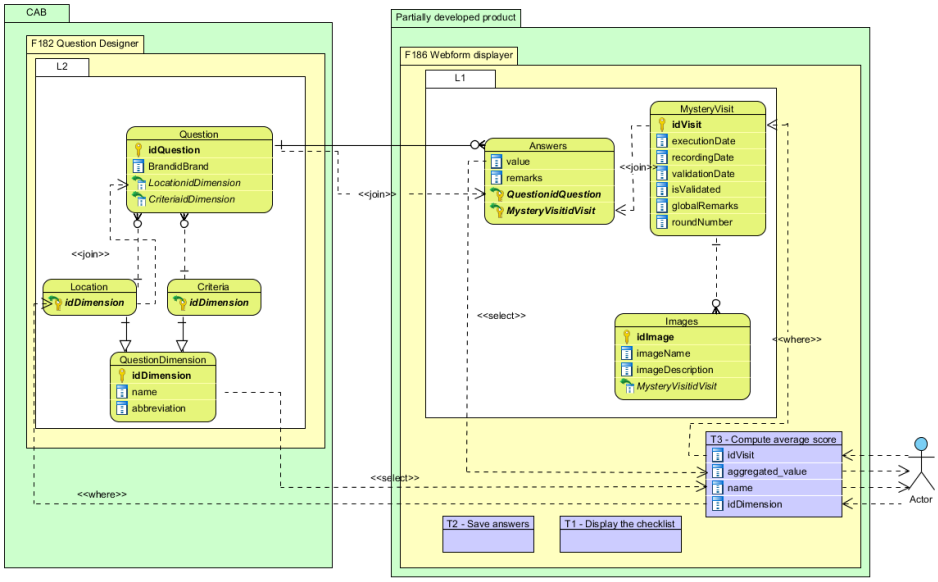


Figure 7.2: Illustration of a product consisting of F182 and F186.

when scoping. Consider two features identified in our example: F186 *Webform Displayer* and F182 *Question Designer*. For each of these two features, the first step is to determine feature's boundary. The feature boundary is defined by what is maintained inside the feature. Once it has been established, ILF and EIF of this feature can be identified. For example, two logical files have been identified regarding these features: L2, the questionnaire file, and L1, the mystery visit file (i.e. answers). It is interesting to note that these features treat the logical file differently. F182 deals only with the questions files L2. Since this feature maintains this logical file, L2 is considered as a ILF for F182. F186 deals with both previously cited files. L1 is considered as an ILF since F186 maintains logical data about answers. However, L2 is regarded by F186 as an EIF because it references (but not maintains) logical data about questions. This situation is described in Fig.7.2 where some DETs are also identified.

Now, let us consider a product p, called *Full Product* which will consist of those two features. This situation is depicted in Fig. 7.2. When counting the function points of p, it would be incorrect to sum the function points of both features since it will count L1 twice: once as ILF for 182 and once as EIF for F186.

In order to count correctly the function points of this product, let's define two functions ILF and EIF. Those functions respectively returns the set of ILF

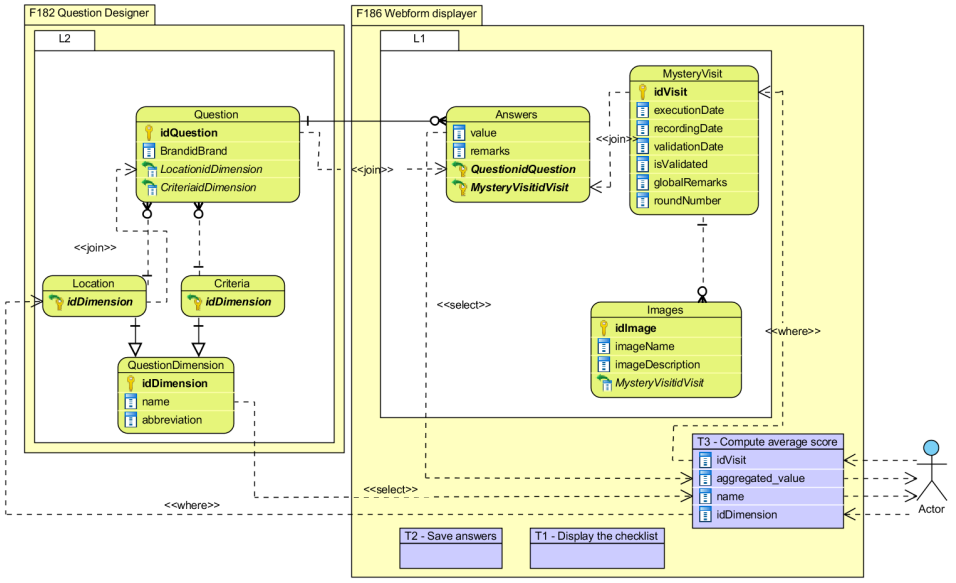


Figure 7.3: Illustration of a product consisting of F182 and F186 but F182 comes from the CAB.

and EIF of a feature. In our example ILF(F182) = {L2} and ILF(F186) = {L1} while EIF(F182) = ∅ and EIF(F186) = {L2}. Consequently, for a given product p, we can define the set of external interface files EIF_p and the set of internal logical file ILF_p as:

$$ILF_p = \bigcup_{f \in p} \mathsf{ILF}(f) \tag{7.3}$$

$$EIF_p = \bigcup_{f \in p} \mathsf{EIF}(f) \setminus ILF_p \tag{7.4}$$

As previously explained, counting function points when dealing with software product lines rises another difficulty. Indeed, the distinction between an ILF and an EIF lies in the fact that the logical file is or not maintained in the application. However, this distinction is no more precise enough for SPL since it consists of a set of products derived from a CAB, having as consequence that some logical files are maintained in the CAB and not in the product (although being part in the product).

Then, a product of the SPL can be seen as the union of two separate applications linked together. The first one consists of the set of features derived from the CAB. Those features have been already implemented and do not need to be

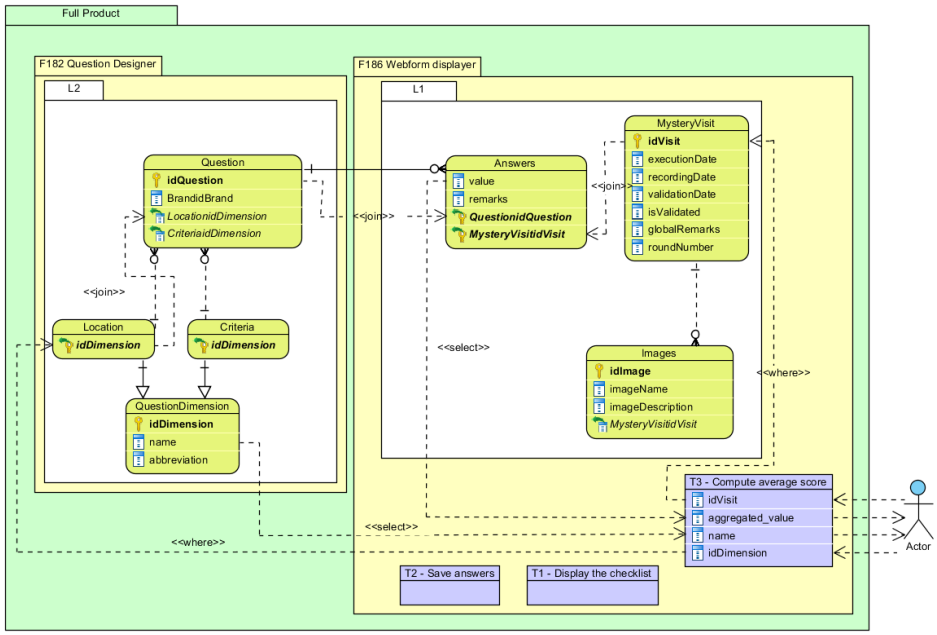


Figure 7.4: Example of features with logical file and DETs

developed again as they can just be reused. The second one is the set of features specifically developed for this product. This is the case in situation depicted in Fig. 7.4 where the product *Partially developed product* relies on the use of a feature developed in the CAB.

Data Functions which are within the boundary of the CAB are referred as Core Logical Filess (CLFs). They have to be considered as already developed outside the boundary of the product. Consequently, as the product will interact with logical file outside its boundary, we have to take them into account as additional EIF. So, the definition of ILF_p and EIF_p have to be redefined to consider CLFs. If we consider that LF_{CAB} is the set of CLFs, then, our functions become:

$$ILF_p = \bigcup_{f \in p} \mathsf{ILF}(f) \setminus LF_{CAB} \tag{7.5}$$

$$EIF_p = \bigcup_{f \in p} \mathsf{EIF}(f) \setminus ILF_p \tag{7.6}$$

This discussion highlights the difficulty to define a product boundary in a SPL Scope context. We see that in contrast with traditional FPC, we need a distinc-

		DET		
		≤ 19	20 to 50	≥ 51
RET	1	Low	Low	Average
	2 to 5	Low	Average	High
	≥ 6	Average	High	High

Table 7.1: Complexity levels for Data Functions

Function Points	ILF	EIF
Low	7	5
Average	10	7
High	15	10

Table 7.2: This table depicts how much function points will be counted for the data functions depending on their complexity level and their DF type

tion between feature boundary, product boundary and the software product line boundary and that the relationships between those boundaries are not direct and can change given a certain scope.

Once ILF and EIF identifications have been processed, we need to estimate the complexity of each of these logical files. It is performed with specific rules which count the number of data element types (DETs) related to each logical file as well as the number of Record Element Typess (RETs). A DET is a unique user recognizable, non-repeated field and a RET is a user recognizable subgroup of data elements within an ILF or EIF [2]. For instance, the logical file L1 contains three RETs (namely Answers,MysteryVisit,Images). Once DETs and RETs have been counted, we can determine the complexity level of the DF (cf. Tab. 7.1). On the basis of this complexity level as well as depending on the Data Function type, we can compute the function points of this logical file (cf. Tab. 7.2). A comprehensive description of all rules to be applied in order to identify DETs and RETs are available in the IFPUG Manual [2].

Transactional Functions

As described in the Function Point Counting Practices Manual provided by the International Function Point Users Group (IFPUG) [2], transactional functions represent the functionality provided to the user for the processing of data by an application. There are three types of transactional functions which are defined in

the manual [2]:

External Inputs: An External Inputs (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

External Outputs: An External Outputs (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information . The processing logic must contain at least one mathematical formula or calculation, create derived data, maintain one or more ILFs or alter the behavior of the system.

External Inquiries: An External Inquiries (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF of EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

The function points contribution of each transactional function is dependent on its complexity. This later is based on the number of File Types Referenced (FTR) by the transactional function and DETs that cross the application boundary during the execution of this transactional function.

In this model, we consider that Transactional Functions are included into unique feature. This means that all FTRs of the Transactional Functions are considered either as an ILF or an EIF of this feature. There is no Transactional Functions that are defined on several features. Nonetheless, we need to determine if EI, EO, and EQ require special treatments regarding previously identified issues: *FP for scoping* and *FP for SPLE* issues. This implies that we have to see how the boundary of the application impacts the Transactional Function contribution and how to interact with the CAB.

The counting of FTR is independent of the file type, i.e. EIF and ILF are both valued as one FTR. However, the counting of DETs depends on the fact those DET cross the application boundary or not. For example, let's consider that the feature F186 contains three Transactional Functions T1 "*Display the checklist*", T2 "*Save answers*" and T3 "*Compute the average score for one question dimension*". Since T1 will use the *name* attribute of *QuestionDimension* in order to display the checklist, it uses a DET that cross the feature boundary, so this DET will be taken it into

account. As a counter example, when saving answers, the *idVisit* attibute does not cross the feature boundary since it internally computed. Then, there no need to count it. Thus, determining the TF contribution will require some considerations.

Let's take an example. Consider the a product p consisting of feature F182 and F186. Let's consider the transactional function T3 "**Compute the average score for one question dimension**" which is of External Output type. It is a TF provided by the feature F182. DETs that enter the feature boundary are:

- *idVisit* which is the id of the visit we want to compute the value,
- *idDimension* which is the id of the dimension which we want to cumpute the value ,
- *name* which is the name of the dimension (provided by F182),
- *idQuestion* which (under the form of a set) collects all id of questions that are related to this specific dimension (it comes from F182).

DETs that leave the feature boundary are:

- *aggregated_value* which is the computed value for the selected dimension and visit,
- *name* which is the name of the dimension (provided to the user),
- *idDimension* which is the id of the dimension for which we want to know the applicable question list.

The total number of crossing DETs is 7. However, this number has to be reduced to 5 since *idDimension* and *name* cross twice the feature boundary. However, this number is based of the feature boundary and not the application boundary. It is correct for the *Partially developed product* from Fig. 7.4 but if you consider the product *Full Product* from Fig. 7.2 consisting of both F186 and F182, the total of DET crossing the product boundary has to be reduced to 4. Indeed, *idQuestion* does not cross the application boundary.

Once the number of DETs and FTRs has been counted, they are used to evaluate the complexity level of this Transactional Function. Relations between complexity levels and number of DETs/FTRs are depicted in Table 7.3 for EI, EO and EQ. The complexity level is then used to determine the number of function points to be considered for each transactional function as depicted in Table 7.4. For instance, T3 in the *Full Product* case has four crossing DETs and two FTRs, because it is an EO, it has a complexity level of *Low* and amounts for four function points.

EI		Boundary crossing DETs		
		≤ 4	5 to 15	≥ 16
FTR	≤ 1	Low	Low	Average
	2	Low	Average	High
	≥ 3	Average	High	High

EO/EQ		Boundary crossing DETs		
		≤ 5	6 to 19	≥ 20
FTR	≤ 1	Low	Low	Average
	2 to 3	Low	Average	High
	≥ 4	Average	High	High

Table 7.3: Complexity level determination for EI, EO and EQ

Function Points	EI	EO	EQ
Low	3	4	3
Average	4	5	4
High	6	7	6

Table 7.4: This table depicts how much function points will be counted for the transactional functions depending on their complexity level and their TF type

7.4 Optimizing the SPL Scope

In this section we describe the mathematical model for finding the optimal scope of a SP (i.e. the scope which minimizes the cost). We showed in Section 6.3 that the requirements engineering problem of SPLs formulated with CORE consists of three steps. First, we have to determine who the relevant customers are (possibly from different domains) and what their needs are. This requires the identification and evaluation of $K_i, T_i \mid\sim G_i, S_i, Q_i$ for different customers i. The second step is defining what the products are constituted of. Answering this question implies that we need to identify a set of features F which can be used to derive product members, i.e. $p \in \mathcal{P}(F)$ and then evaluate them in terms of data and transactional functions. Thirdly, we need to identify conditions for the product to realize the tasks, i.e. K'. In order to describe the model, we start from the MIP 3.1 and then focus on the description of features and function points. The whole set of indexes, parameters and decision variables are described in Tab. 7.5 and Tab. 7.6.

Table 7.5: Description of indices and parameters.

Indices	
m	goal models $m = (1,\ldots,M)$
p	the software products $p = (1,\ldots,P)$
k	features $k = (1,\ldots,K)$
y	the software product line members, i.e. the products and the CAB $y = (1,\ldots,M,\Omega)$
j	the logical files $j = (1,\ldots,J)$
t	the transactional functions $t = (1,\ldots,T)$
Parameters	
π_{a_m}	revenue from goal a_m
α_j^I	is the function point contribution of the logical file j considered as an ILF
α_j^E	is the function point contribution of the logical file j considered as an EIF
δ_t^L	is the function point contribution of transactional function t if it reaches at least the low complexity level
δ_t^A	is the function point contribution of transactional function t if it reaches at least the average complexity level
δ_t^H	is the function point contribution of transactional function t if it reaches the high complexity level
$DET_{t,j}$	is the number of DET entering and going out of the logical file j in the transactional function t
$\overline{DET}_t$	is the number of DETs which do not cross the boundary of the feature to or from another feature logical file
FTR_t	is the number of FTR to be counted for the transactional function t

Table 7.6: Description of decision variables.

Decision variables	
Goal Model	
$g_{a_m,t}$	is equal to 1 if the goal a from goal model m with $a_m = 1,\ldots,A_m$ is satisfied during the period t, it is 0 otherwise
$i_{b_m,t}$	is equal to 1 if the inference node b from goal model m with $b_m = 1,\ldots,B_m$ is satisfied during the period t, it is 0 otherwise
$t_{c_m,t}$	is equal to 1 if the task c from goal model m with $c_m = 1,\ldots,C_m$ is realized during the period t, it is 0 otherwise
Software Product Line	
Decision variables - Product Composition	
s_p	is equal to 1 if the product p has been developed, it is 0 otherwise
$f_{k,p}$	is equal to 1 if the feature k has been integrated in the product p, it is 0 otherwise
Decision variables - Data Functions	
$iLF_{j,y}$	is equal to 1 if the logical file j has to be maintained in the SPL as an ILF or CFL, it is 0 otherwise
$ILF_{j,y}$	is equal to 1 if the logical file j has been counted in the member y of the SPL as ILF, it is 0 otherwise
$EIF_{j,y}$	is equal to 1 if the logical file j has been counted in the member y of the SPL as EIF, it is 0 otherwise
$LF_{j,y}$	is equal to 1 if the logical file j has been counted in the member y as ILF, EIF or CFL, it is 0 otherwise
Decision variables - Transactional Functions	
$TF_{t,y}$	is equal to 1 if the transactional function t is used in member y
$TF_{t,y}^{DET}$	is the number of DET in the transactional function t for family member y
$TF_{t,y,1}^{CF}$	is equal to 1 if the number of DETs of the transactional function t used in member y contribute at least to 1 in the complexity factor, it is 0 otherwise
$TF_{t,y,2}^{CF}$	is equal to 1 if the number of DETs of the transactional function t used in member y contribute to 2 in the complexity factor, it is 0 otherwise
$TF_{t,y}^{Cpl}$	is the complexity level of the transactional function t for family member y
$TF_{t,y}^{L}$	is equal to 1 if the transactional function t is at least of low complexity, it is 0 otherwise
$TF_{t,y}^{A}$	is equal to 1 if the transactional function t is at least of average complexity, it is 0 otherwise
$TF_{t,y}^{H}$	is equal to 1 if the transactional function t is of high complexity, it is 0 otherwise
$TF_{t,j,y}^{EIF}$	is equal to 1 if the logical file j has to be counted as an EIF for the transactional function t in family member y

The Product Line

The following equation is the objective function minimizing the cost of the whole SPL Portfolio.

$$\text{minimize :}$$

$$w\left(\sum_p \sum_j \left(\alpha_j^I ILF_{j,p} + \alpha_j^E EIF_{j,p}\right)\right) \tag{7.7}$$

$$+w\left(\sum_p \sum_t \left(\delta_t^L TF_{t,p}^L + \delta_t^A TF_{t,p}^A + \delta_t^H TF_{t,p}^H\right)\right) \tag{7.8}$$

$$+\mathbf{w}\left(\sum_j \left(\alpha_j^I ILF_{j,\Omega} + \alpha_j^E EIF_{j,\Omega}\right)\right) \tag{7.9}$$

$$+\mathbf{w}\left(\sum_t \left(\delta_t^L TF_{t,\Omega}^L + \delta_t^A TF_{t,\Omega}^A + \delta_t^H TF_{t,\Omega}^H\right)\right) \tag{7.10}$$

The cost consists of the effort deployed to implement both data functions and transactional functions as well for each product as for the CAB. Briefly speaking, it consists of the sum of function points multiplied by a *cost per function point* factor, w or $\mathbf{w}$. The former (w) is a cost factor for normal development, the latter ($\mathbf{w}$) is a cost factor for reusable development. Terms (7.7) and (7.8) are dedicated to products development while terms (7.9) and (7.10) are related to the CAB.

Terms (7.7) and (7.9) are focused on data functions. For each SPL member y (i.e. the CAB is considered as a SPL member), this term sums the contribution of each logical file j depending if it is considered in this product member as an ILF or EIF (or not included at all). α_j^I and α_j^E are the function points of data types. Possible values are depicted in Table 7.2. As discussed in section 7.3, the data type complexity of each logical file is not depending on the scoping process. It is considered as a parameter in our model and it has to be computed for each logical file prior to the optimization.

Terms (7.8) and (7.10) are focused on transactional functions. For each SPL member y, we sum the contribution in terms of function points of each transactional function. Depending on product boundaries, each transactional function can reach a low (δ_t^L), average (δ_t^A) or high level (δ_t^H) of complexity. This complexity level will determine the number of function points to be counted. Possible values for δ_t^L, δ_t^A and δ_t^H are described in Tab.7.4.

Data Functions

For each product, we need to ensure that logical files related to the feature composition of this product have been counted. These logical files can be ILFs (internal to the product boundary) or EIFs (external to the product boundary) of the product as well as coming from the CAB. If we consider a product p with a set of features, all ILFs of this set of features (i.e. within the feature boundary) have to

be internal to the product boundary (or they can also come from the CAB). For that, we need to introduce a new type of logical file, let's talk about internal product logical file (*iLF*). The set of *iLFs* for a product is the union of all the ILFs of the product features. From the point of view of the product-to-be, each internal product logical file could be implemented as an ILF of that product or as an ILF of the CAB. The latter implies that if the ILF is developed within the CAB, a related interface, i.e. an EIF, would be developed in the considered product. We have for each product that:

$$\forall p,k : f_{k,p} \leq \frac{1}{|\mathsf{ILF}(k)|} \sum_{a \in \mathsf{ILF}(k)} iLF_{a,p} + \frac{1}{|\mathsf{EIF}(k)|} \sum_{b \in \mathsf{EIF}(k)} LF_{b,p} \tag{7.11}$$

This constraint states that if a feature is used in a product, then all ILF of this feature has to be an *iLF* of the product (i.e. it can be an ILF of the product or an ILF of the CAB) and all EIF of this feature has to be considered in the product (potentially as ILF, EIF or ILF of the CAB since at this stage, we do not know if combined use of features will transform some EIF into ILF). Indeed, consider a product consisting of features f_1 and f_2. lf_1, a logical file, is an EIF for f_1 but an ILF for f_2. Due to f_2 it will be integrated as an ILF of either the product or the CAB even if it was only a EIF for f_1.

If a logical file j for product p was considered previously as an iLF, it means that this logical file will be either an ILF of this product or it has to be developed as an ILF of the CAB (member Ω of the software family) with its related interface for the product p, i.e. $EIF_{j,p}$. It is stated in the following constraints:

$$\forall p,j : iLF_{j,p} \leq ILF_{j,p} + \frac{(ILF_{j,\Omega} + EIF_{j,p})}{2} \tag{7.12}$$

Then, we have to constraint that for each member family y, the set of LF of this member, identified in equation 7.11 (but, as we will see later, also in equation 7.24), can be either an EIF for the product or an ILF:

$$\forall j,y : LF_{j,y} \leq ILF_{j,y} + ELF_{j,y} \tag{7.13}$$

Transactional Functions

Then, we have to determine which transactional functions will be used in our products. As we said previously, transactional functions are related to features. For each feature k included in a product p, we have to ensure that the related transactional functions of this feature have been included.

Assume that for each feature k, the set of required transactional functions for this feature is given by T_k. Then, we have the following set of constraints:

$$\forall p,k : f_{k,p} \leq \frac{1}{T_k} \sum_{t \in T_k} (TF_{t,p} + TF_{t,\Omega}) \tag{7.14}$$

This set of equations states that the transactional functions related to feature k can be either developed specifically for the product (i.e. $TF_{t,p}$) or derived from the CAB (i.e. $TF_{t,\Omega}$).

However, we have to state that if a transactional function has been developed in the CAB, it is no more necessary to develop it in the product.

$$\forall p,k : TF_{t,p} + TF_{t,\Omega} \leq 1 \tag{7.15}$$

Once we know if a TF is used either from the CAB or not, we need to know how many Function Points will be counted for each Transactional Function. As previously discussed, it depends on the number of DETs crossing the product boundary. Crossing the product boundary means that the transactional function use DETs coming from or going out an external logical file of the product. In order to take that into account, we have to know if a logical file is internal or external to the product. If it is external, then we need to count DETs that will be used from this file or manipulated from it. This is stated in the following set of equations. Consider that FTR_t is the set of logical files manipulated by the transactional function t.

$$\forall l \in FTR_t : TF^{EIF}_{t,l,y} + 1 \geq TF_{t,y} + EIF_{l,y} \tag{7.16}$$

This set of equations states that if a transactional function t is used in a product member y (i.e. $TF_{t,y}$ is equal to 1) and that simultaniously the logical file l is an EIF for this member (i.e. $EIF_{l,y}$ is equal to 1), then the decision variable $TF^{EIF}_{t,l,y}$ has to be set to 1 in order to satisfy the inequality. This latter variable models that DETs related to the logical file l for the transactional function t in member y will have to be counted during the function point counting.

So, to determine the number of DETs of a transactional functions t in a product member y, noted $TF^{DET}_{t,y}$, we count all the DETs of this TF which cross the member boundary. In order to do that, we use the decision variable $TF^{EIF}_{t,l,y}$ defined in equations 7.16. This variable allows to know if DETs used in the transactional function t and related to the logical file l has to be taken into account. This number of DETs is recorder in the parameter $DET_{t,j}$. We also need to count DETs that are not directly related to specifical logical files. For example, some outputs of the transactional functions which will be given to the user. The *aggregated_value* from Fig.7.3 is such an example. These DETs are recorded into the parameter $\overline{DET}_t$. The whole computation of DETs are described in the following equations.

			1 to 5 DET	6 to 19 DET	20 or more DET
		Points	0	1	2
FTR	0 to 1	0	0	1	2
	2 to 3	1	1	2	3
	4 or more	2	2	3	4

Table 7.7: Complexity Factors for EO and EQ.

$$\forall t, y : TF_{t,y}^{DET} = \sum_{j \in J} DET_{t,j} * TF_{t,j,y}^{EIF} + \overline{DET}_t * TF_{t,y} \tag{7.17}$$

Once the number of DETs have been counted, we need to know the complexity level of this transactional function. As explained in section 7.3, it depends on the complexity rating table which return a complexity level for particular numbers of DETs and FTRs (see Tab.7.3). In order to calculate that, we use an intermediary value called the complexity factor. This factor is used to map the numbers of DETs and FTRs to the complexity level. The complexity factor of a TF is directly computed on the number of its DETs and FTRs. This relation is depicted in table 7.7. For example, if a EO handles 6 to 19 DETs, it gets 1 point, 0 if it is less and 2 if it is more. Regarding the FTR dimension, if this EO handles 2 to 3 FTRs, it gets 1 point, 0 if it is less and 2 if it is more. Then we consider that EO with 1 or less point are of low complexity. If it is evaluated at 2 points, it is of average complexity. More than 2 points, means that the EO is highly complex.

Given the value of this complexity factor, we can determine if the transactional function has a low, average or high complexity level. To determine the contribution of DETs to the complexity factor, we use the following set of equations:

$$\forall t, y : TF_{t,y,1}^{CF} \geq \frac{TF_{t,y}^{DET} - a}{9999} \tag{7.18}$$

$$\forall t, y : TF_{t,y,2}^{CF} \geq \frac{TF_{t,y}^{DET} - b}{9999} \tag{7.19}$$

Where a is equal to the number max of DETs to contribute to zero and b is the number max of DETs to contribute to 1. a and b depends on the type of transactional function considered as described in Table 7.3. $TF_{t,y,1}^{CF}$ will be equal to 1 if the number of DETs is enough to contribute to 1 point in the complexity

factor. $TF^{CF}_{t,y,1}$ and $TF^{CF}_{t,y,2}$ will be both equal to 1 if the number of DETs contribute to 2 in the complexity factor.

Equation 7.20 describes the calculation of the complexity factor $TF^{Cpl}_{t,y}$ for transactional function t from product member y. It sums the contribution of the DETs and the FTRs of this transactional function. As the number of file type referred, TF^{FTR}_{t}, is independent of the product member boundary, it is considered as a parameter of the model.

$$\forall t, y : TF^{Cpl}_{t,y} \geq TF^{CF}_{t,y,1} + TF^{CF}_{t,y,2} + TF^{FTR}_{t} * TF_{t,y} \tag{7.20}$$

Once, the complexity factor $TF^{Cpl}_{t,y}$ has been calculated, we can determine if the transactional function is at least low, average or high level. It is determine with the following 3 sets of equations:

$$\forall t, p : TF^{L}_{t,p} * 9999 \geq TF^{Cpl}_{t,p} \tag{7.21}$$

$$\forall t, p : TF^{A}_{t,p} * 9999 \geq TF^{Cpl}_{t,p} - 2 \tag{7.22}$$

$$\forall t, p : TF^{H}_{t,p} * 9999 \geq TF^{Cpl}_{t,p} - 3 \tag{7.23}$$

It means that if the transactional function t is of high complexity level, the three decision variables $TF^{L}_{t,p}$, $TF^{A}_{t,p}$ and $TF^{H}_{t,p}$ will be set to 1. The subtraction by 2 and 3 for the second and third sets of equation 7.23 represent the upper bound of the average and high complexity level of the complexity factor (see Tab.7.7).

Finally, we have to take into account that a transactional function could be developed as part of the CAB. However, it is not sure that all logical files related to this feature would be incorporated into the CAB. So, we need to ensure that if a transactional function has been incorporated, all logical files referred by this transactional function have been at least included as EIF in the CAB. This is stated by the following set of equations:

$$\forall l \in L_t : TF_{t,\Omega} \leq \frac{1}{|L_t|} \sum_{l \in L_t} (LF_{l,\Omega}) \tag{7.24}$$

Due to the set of equations 7.13, it is ensured that a LF decision variable stated to 1 will result into an ILF or an EIF in the CAB.

7.5 Application on an Example

In this section, we apply our method to the case study of *market maker* Software AG [35, 101]. In the first part of this section, we present the organization and we

state both problem and mathematical model. In the second part, we present the result of the SPL scope optimization.

Problem Statement

market maker is an organization providing applications able to collect, display and manage financial data. At the end of 2004, the SPL development team consisted of around 25 people and the annual revenue was €5 million per year. For this organization, SPLE was regarded as a key strategic element in addressing new market segments. In 1999, "*when markets were boiling and the demand for innovative products was immense*", they started the development of a SPL for web-based applications, called i*ProductLine nearly from scratch. This SPL aimed at developing web-applications able to collect, validate, store, analyze, aggregate, repackage and distribute financial data. We show in the remainder of this section how our mathematical model can be applied to the market marker case study and how it can determine the SPL scope.

In 2004, i*ProductLine was instantiated for various markets: information systems for asset managers in banks, market data servers integrated in brokerage systems for on-line ordering, specialized data display services for metal traders and grains and oilseed traders and, content provisioning for financial web portals. A small goal model is depicted for each market segment in Fig.7.5. For the rest of this example, we assume some potential revenue for each satisfied goal. Those revenues are presented in thousands of Euro near each related goal. Mandatory goals have bold circles.

On the basis of the case study description, we identified a set of 11 grained features able to realize tasks identified in each goal models. They are described in Tab.7.8 and their relations with tasks are described in Tab.7.11. In this example, we did not consider legacy systems. Some feature dependencies were identified before the optimization of the product portfolio. The feature "*Assets management*" requires the "*User management*" feature to work. Two other constraints stated that both "*Stock Analysis*" and "*Display Stock in Table*" required the feature "*Store Stock Data*". In the same way, "*Store Historical Commodities Data*" was required for product members using "*Display Commodities Data in Table*". Those constraints were added into the mathematical model.

Once, these features and their constraints were identified, we determined different data and transactional functions for each features: 8 logical files which are briefly depicted in Tab.7.9 and 14 transactional functions which are presented in Tab.7.10. For each logical files and transactional functions, we determined sets of DETs and incorporated them into the mathematical model.

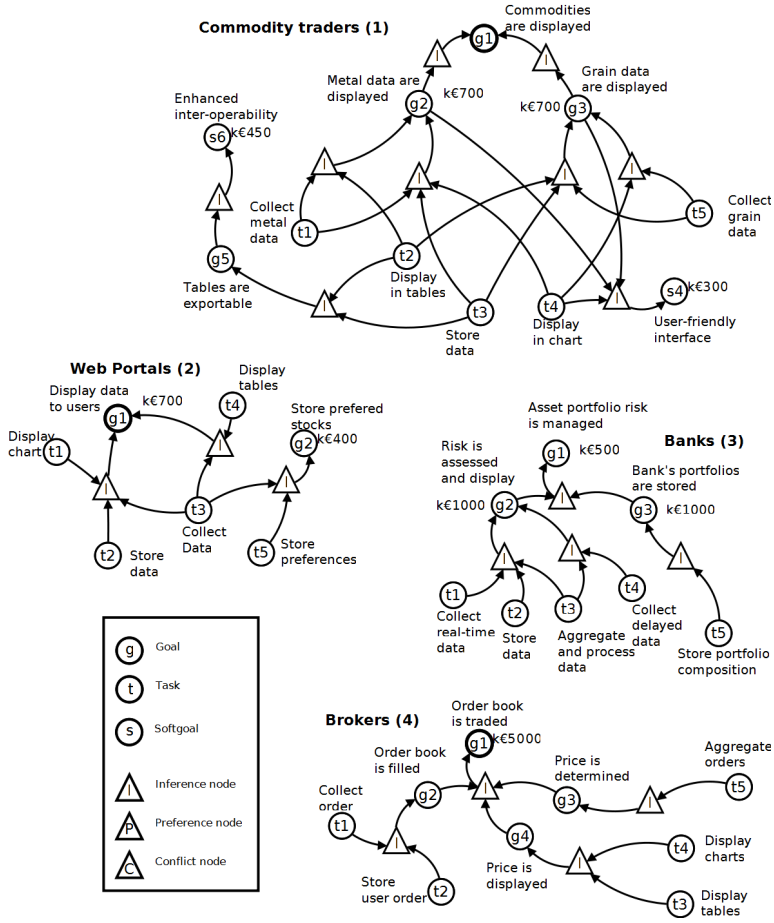


Figure 7.5: The 4 goal models used to design *market maker* customer segments

	Feature	ILF	EIF
f_1	Assets management	lf_2	lf_1, lf_3
f_2	Collect real-time stocks data		lf_1
f_3	Orders placement	lf_5	lf_1
f_4	User management	lf_3	
f_5	Stock Analysis		lf_1, lf_6
f_6	Orders execution	lf_5	lf_3
f_7	Collect real-time commodities data		lf_7
f_8	Display stock data in table		lf_6
f_9	Display commodities data in table		lf_8
f_{10}	Store historical commodities data	lf_8	
f_{11}	Store stocks data	lf_6	

Table 7.8: Description of identified features and the related logical files

LF	File Name	Examples of DETs
lf_1	Stocks real-time data	stock_id, stock_price...
lf_2	Portfolio	stock_id, customer_id...
lf_3	Users	customer_id, customer_name...
lf_5	Orders	order_type, quantity, bid_price, ask_price...
lf_6	Stock historical data	stock_id, price, date
lf_7	Commodities real-time	commodity_id, commodity_current_price...
lf_8	Commodities historical data	commodity_id, price, date

Table 7.9: Logical files identified on the basis of the features

Once goals and features were identified and evaluated, we had to express how those features could realize the tasks from the goal models. Such assessment is depicted in Tab.7.11.

Results Presentation

The entire model was written and resolved with the GNU Linear Programming Kit[2] (GLPK). The model is partially implemented in AnalyticGraph. The composition of each product in terms of features is depicted in Tab.7.12. An advanced analysis of commonalities and variabilities in terms of features resulted in the feature diagrams illustrated in Fig.7.7. Since they were always used together in product 2 and 4, features f_1, f_2 and f_4 were aggregated in a compound feature called "portfolio management". We also defined two other compound features called

[2]http://www.gnu.org/software/glpk/

TF	Trans. Funct. Name	Feature	FTR	TF Type
tf_1	buy_assets()	f_1	lf_1,lf_2, lf_3	EI
tf_2	show_assets()	f_1	lf_2, lf_3	EQ
tf_3	sell_assets()	f_1	lf_1, lf_2, lf_3	EI
tf_4	collect_stocks_data()	f_2	lf_1	EO
tf_5	display_stock_data()	f_8	lf_6	EO
tf_6	add_order()	f_3	lf_1,lf_5	EI
tf_7	delete_order()	f_3	lf_5	EI
tf_8	add_user()	f_4	lf_3	EI
tf_9	delete_user()	f_4	lf_3	EI
tf_{10}	beta()	f_5	lf_6	EQ
tf_{11}	growth	f_5	lf_1,lf_6	EQ
tf_{12}	process_order()	f_6	lf_5	EI
tf_{13}	collect_commodities_data()	f_7	lf_7	EO
tf_{14}	display_commodities_data()	f_9	lf_8	EO

Table 7.10: Description of the Transactional Functions with their related feature, their File Types Referred and their type

Web Portals	Commodity Traders	Brokers	Banks
$(f_8 \rightarrow t_1)$	$(f_7 \rightarrow t_1)$	$(f_2 \rightarrow t_1)$	$(f_3 \rightarrow t_1)$
$(f_{11} \rightarrow t_2)$	$(f_9 \rightarrow t_2)$	$(f_{11} \rightarrow t_2)$	$(f_4 \rightarrow t_2)$
$(f_2 \rightarrow t_3)$	$(f_{10} \rightarrow t_3)$	$(f_5 \rightarrow t_3)$	$((f_8 \wedge f_{11}) \rightarrow t_3)$
$(f_8 \rightarrow t_4)$	$(f_9 \rightarrow t_4)$	$(f_{11} \rightarrow t_4)$	$((f_9 \wedge f_{10}) \rightarrow t_4)$
$((f_4 \wedge f_1) \rightarrow t_5)$	$(f_7 \rightarrow t_5)$	$((f_4 \wedge f_1) \rightarrow t_5)$	$(f_6 \rightarrow t_5)$

Table 7.11: Task realization description

"Brokers module" and "Commodities module" which respectivelly consisted of f_6, f_9, f_{10} and f_7, f_9, f_{10}.

Fig.7.6 presents the composition of the CAB as well as each product in terms of data and transactional functions. The Core Asset Base, i.e. the set of software artifacts that are used in the production of more than one product in a SPL [35], consisted of four internal logical files. It also included two external interfaces related to logical files maintaining real-time information about stock and commodities data. A large set of transactional functions were integrated in the CAB. Each data and transactional functions implemented specifically for each product are also depicted in Fig.7.6

Markets	Satisfied goals	Feature composition
Commodity Traders	$g_1, g_2, g_3, g_4, g_5, g_6$	f_7, f_9, f_{10}
Web Portals	g_1, g_2	$f_1, f_2, f_4, f_8, f_{11}$
Brokers	g_1, g_2, g_3	$f_6, f_8, f_9, f_{10}, f_{11}$
Banks	g_1, g_2, g_3, g_4	$f_1, f_2, f_4, f_5, f_{11}$

Table 7.12: Solution description in terms of satisfied goals and delivered features

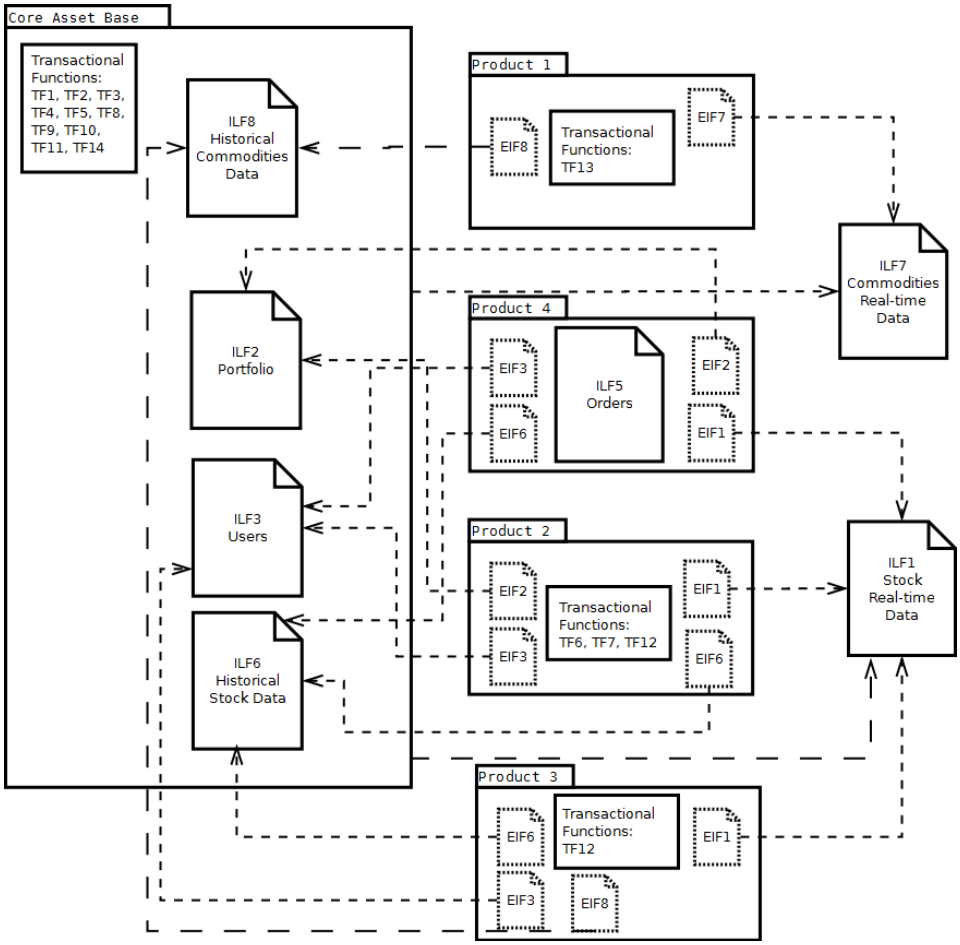


Figure 7.6: Schematic description of the resolution of the example

7.6 Related Work

Although several SPL scoping methods have been designed, few address the optimization goal of the product portfolio scoping on the basis of the profitability concern.

Recently, J. Müller suggested a mathematical program to optimize the product portfolio on the basis of a profit maximization objective [106]. His model is based on previous work in product lines (i.e. not software) and on conjoint analysis. Although the idea to use a mathematical program to optimize the SPL scope is similar to ours, the underlying assumptions of the two models are different. First, he assumes a SPL and all its valid configuration products and it optimizes the selection of the product-segment pair given that a configuration products can be implemented by a various set of asset components. His method allows to determine the optimal price and take competitors into account. However, it does not introduce any time considerations what prevents any possibility of setting development priorities or release planning which is however a primary concern in SPLE and assume an a priori complete SPL specification.

Several survey of Software Product Line Scoping have been performed. In their survey [79], I. John and M. Eisenbarth identify 16 scoping approaches but they highlight that scoping optimization has only been partially addressed. Moreover none of them integrate the optimization of the three types of scoping.

Regarding the five methods addressing the product portfolio analysis [27, 74, 117, 140, 142] only [27] also addresses asset scoping. Moreover as highlighted by J. Müller none of those five methods integrate market and cost perspectives which prevents any profitability consideration.

The Quality Function Deployment Product Portfolio Planning suggested by Helferich [74] elicits required features of products of a SPL and asks engineers about technical feasibility. It also allows to identify customer segments. However it does not consider revenue and cost aspects and therefore cannot be used for profit maximization. However, this method can usefully be regarded as complementary to ours since it identifies features and can be used to match them with goal models.

Niehaus et al. present a Kano model which allows to design customer-oriented SPL [117]. It is focused on the product portfolio planning but does not consider any asset scoping.

The issue of a SPL release planning is addressed by several authors but they do not address neither composition of product portfolio nor asset scoping [140, 142]. In single software engineering, Denne et al. already highlighted the importance of "*optimizing the time at which value is returned to the customer, instead of concentrating only on controlling risk and cost*. [43]" They suggest a method to decompose the

system into Minimum Marketable Features (i.e. units of customer-valued functionality) and determine the sequence of incremental release which optimizes the NPV of the system. Their research results are complementary to our method.

7.7 Conclusion and Further Work

In this paper we suggested that when scoping, identification and evaluation of the three scope types (i.e. product portfolio, domain and assets scoping) can be performed separately but the optimization of those scopes has to be performed in an integrated model. Consequently, we proposed a mathematical model able to optimize the SPL scope. Our method is based on the assumption that all SPL objectives are eventually driven by profit maximization. Revenue is a function of the customer satisfied needs and cost is a function of the feature development effort which is estimated with a function point analysis.

Our model is based on the description of customer needs in terms of goals. This use is justified because first, we showed that goals are better than features to capture the customer's WTP and secondly, they are able to capture the alternative solutions of a requirements problem what is useful to find large commonalities between market segments. More precisely, we use Techne because it allows automated reasoning and it is compatible with other goal-oriented requirements language.

We also provided a method to use function point analysis during SPL Scoping. We identify two issues related with such approach, namely the *FPC for scoping issue* and the *FPC for SPLE issue*. We showed how to address those issues in our model.

The output of our model is a good basis for further commonalities and variabilities analysis. We also demonstrated its applicability with an example based on the *market maker* case study.

Regarding limitations of this approach, aside from the global limitations of ROP (discussed in Chapter 4), we should highlight that it is difficult to estimate the cost factor.

In further work we will consider the following extensions. First, we estimated revenue on the WTP of satisfied goals. However, this assumption constraints the applicability of our model to monopoly setting. Further work should integrate customer decision in presence of competitors products. Secondly, we will study implications and difficulties of using conjoint analysis with goals instead of features. Thirdly, we limited our definition of features to Batory's definition which allows to simplify the relation between feature and data or transactional functions. However, a more complete feature definition would require to take into account the non-monotonicity between features and SPL cost function. Then, we

need to integrate in our model both risk management (e.g. with stochastic models) and maintenance cost which should impact the CAB size and legacy components integration.

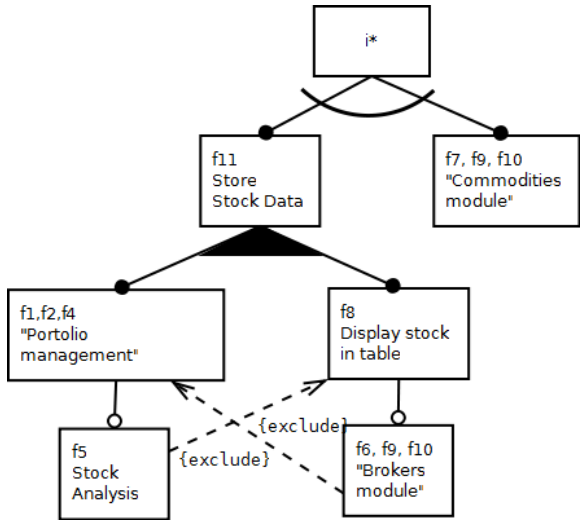


Figure 7.7: A possible feature diagram for i*ProductLine

8 Optimization of Self-Configurable Systems

8.1 Introduction

Context: Evolution of Requirements

Software requirements can change over time, as can the organizational and technological environment in which the software runs. It was well illustrated in the MystShop case where requirements never stop to change. Consequently, it is hard to get the requirements right at the beginning of a project. In order to remain relevant when these changes happen, we say that software evolves in response. Software evolution, and more specifically its management, is an important challenge which has been recognized for a long time in software engineering and requirements engineering research.

Living with software evolution involves also dealing with changes of requirements, or the so-called Requirements Evolution Problem, as they drive to some extent software evolution in the first place [49]. The crux of the problem is how to deal with the fact that we know at design time that requirements will change at run time, yet we do not know exactly how they will change. There are essentially three complementary ways to address the Requirements Evolution Problem: (i) to try to anticipate as much as feasible the future changes of requirements and of the environment, and design features into the system which will satisfy these future requirements; (ii) to build sensors and feedback loops into the system, so as to monitor requirements satisfaction and in case of failure for example, to activate and deactivate some features – this is roughly speaking the adaptive systems approach; (iii) to identify versatile modular and configurable features, which users can combine in ways which cannot be anticipated today, or in other words, to give users a toolset, which they can learn to configure and work with on their own, in ways which will satisfy their future requirements.

Scope: Self-Configurable Systems

In this chapter, we are interested in the latter approach to deal with requirements changes. We observed it in the MystShop case when administrators wanted new

checklists to be used as proof-of-concept for prospects. At this time, Mystery Shoppers ask some configurable features in order to adapt the system to their needs. We call such systems Self-Configurable Systems (SCS). The basic idea is that the system engineers will not produce a system that satisfies all the specific requirements that the various stakeholders may have. For the MystShop case, it would be impossible to anticipate all possible checklists. Instead, they will engineer a system which can, if properly configured *by the users*, satisfy these requirements, whichever they are at system design time, and whichever they may end up being at run-time. If you think of requirements as being represented, as usual in goal-oriented requirements engineering [40, 144], as a goal forest, where goals close to roots are abstract and general requirements, and goals close to leaves are more specific requirements, then Self-Configurable Systems are engineered to satisfy some goals midway in the goal forest, between the leaves and the roots, rather than satisfy the leaves. Indeed, Ernst et al. [49] suggested that the stability of a requirement over time depends, among others, on the abstraction level of that requirement. A requirement such as "A message should be sent to the relevant person, when a new order arrives" is less likely to change relative to "An email should be sent to the order manager, when a new order arrives". As we will see later, this view is simplistic, but it gives an initial idea that we build from in the rest of the chapter.

Although it does seem desirable to want to engineer systems which satisfy abstract requirements, we still have to make sure that the system can satisfy concrete requirements. But instead of designing monolithic system features to satisfy abstract goals, we suggest that potentially many modular, reusable, and configurable features can be engineered in such a way that their different combinations can satisfy both the abstract and the various concrete requirements. The interesting consequence is that *there can be many requirements which such sets of features can be used to satisfy, even if we actually did not anticipate these requirements at design time*. In other words, our approach is akin to seeing features as Lego bricks, which can be combined to make various structures, many if not most of which are actually not anticipated at design time. We called the underlying features Self-Configurable Features (SCF).

Giving configurable features to users can alleviate some run time changes, by transferring part of the design to users and at run time. It could also reduce the time required to match the system to new requirements since it shortens time from a new requirement to a configuration of SCFs which satisfies it.

That said, it is not possible to strictly partition systems into SCS or non-SCS. A system is more or less a SCS depending on the proportion of its requirements that are operationalized with Self-Configurable Features.

SCS are not a new class of systems. In Business Intelligence, users may initially

need dashboards about sales, then they will ask for visualizations of inventory data, later about purchasing, and so on. We cannot anticipate exactly what will be the content of each dashboard, but it can be anticipated that they will reuse such standard data visualizations as pie charts, bar charts, tables, and so on, and that they will involve summing, averaging, or other common computations over data. With Self-Service Systems for Business Intelligence, vendors attempt to provide users with configurable features to design their own dashboards [78]. PowerBI or QlickView are typical examples.

Websites are also systems which can frequently evolve. Some changes are by nature always similar and can consequently be anticipated. For example, a new page must be added, some images need to be inserted, switch in a menu must be done. Since these changes can be frequent, Content Management Systems, such as Joomla[1] or Wordpress[2], have been developed in order to let users make those changes, with minimal intervention of engineers.

Problem: Finding the Optimal Mix of self-configurable features

Providing many configuration possibilities can lead to systems with numerous features, and hence to relatively complex Self-Configurable Systems. By complex, we mean that they provide many features to end-users. The problem with the complexity of a SCS is that it can be negatively correlated with its usability: end-users, who have relatively low (sometimes no) IT background, may not be capable of understanding and combining the features consistently to build custom solutions. That problem of complexity is typical of Self-Configurable Systems: regular systems avoid such complexity by directly satisfying the user specific requirements.

So, providing Self-Configurable Features to users can fail to produce the expected benefits. First, it requires the user to do at run time a part of work which would have normally been done at design time by others, to figure out how to combine and configure features to satisfy her concrete requirements. Secondly, building a system which is configurable, and organized around modular features, is a challenge in itself.

One of the main difficulties when dealing with SCS is, then, to find the right balance between requirements which should be satisfied with Self-Configurable Features, as opposed to other requirements.

For clarity, consider the following situation: a company sells product p. This product is manufactured from 2 sub-products: p_A and p_B. Producing one unit of p requires one unit of p_A and one unit of p_B. Now, let's assume that the company

[1] https://www.joomla.org/
[2] https://wordpress.org/

has 2 employees x and y. Both work 40 hours per week. x can produce in one hour, one unit of p_A or of p_B. For y, both tasks are more complex, which gives longer production times: 2 hours for p_A and 4 hours for p_B. The optimal solution of this problem shows that the optimal allocation asks y to produce mainly p_A because it is in this task that he has a comparative advantage[3].

Contribution

The aim in this chapter is to two-fold. In a first part, we look at the theoretical issues that arise in relation to SCS, how it relates to research in Requirements Engineering (RE), and what new issues, if any, it gives rise to. We first motivate the use of SCS in Section 8.2 and 8.3. We then provide a more detailed definition of what SCS are, with examples in Section 8.4. We discuss how requirements from traditional systems differ from those from SCS in Section 8.5. Finally, we suggest that designing a SCS raises new challenges for RE in Section 8.6. We conclude this first part with a discussion about tradeoffs and possible future works respectively in Section 8.7.

In the second part of the chapter, we explore one of the suggested directions. We suggest to resolve the trade-off problem by formulating it as an optimization problem. This gives three additional contributions to the theoretical part. First, the problem is modeled as a goal-oriented requirement problem. We use Techne as Requirement Modeling Language (RML) and suggest some extensions to the language in order to model problem instances. Secondly, we suggest a mapping between the goal model and an optimization problem which formulates the trade-off problem. Resolving this optimization problem allows to determine the optimal set of requirements that should be treated as satisfiable via Self-Configurable Features, or other features. Thirdly, we present extension made to AnalyticGraph in order to support the approach. All contributions are illustrated with the MystShop case study.

8.2 Illustration - Self-Configurable in Business Intelligence

Although we observed SCS for the MystShop case, it has been approached in different domains. For example, we observed Self-Service Business Intelligence (SSBI). In general, IS for BI gather business data in order to provide information to business decision-makers, under the form of reports, dashboards, or any other relevant output [63, 111]. Despite there being frameworks for doing RE of BI systems [119, 24], there is a practical difficulty that is hard to overcome in terms of

[3]Namely, x makes 10 p_A and 30 p_B while y manufactures 40 p_A.

methodology. It can be formulated as follows: the data types and sources, and the information relevant for business decision-making may not be the same at all times, which means that, for example, the content of reports, the analyses applied to data, and so on, need to be changed regularly. In practice, it may be too costly (or take too much time) to have business analysts elicit new requirements on reports and analyses, and propagate these new requirements through the specifications, system architects to architecture, and software engineers to code each time a change occurs.

From the standpoint of those who make and sell BI systems, it may also be more interesting to avoid changing the systems so much, because, for example, it means maintaining systems that become different from each other over time, even if they started from the same set of functionality. Self-Configurable Systems are a response to this, in that they do not try to satisfy the most specific requirements, but give features whose combinations could satisfy various specific requirements. For example, a Self-Configurable BI (SSBI) system will have features that allow its users (they can include business analysts as well, not only end-users) to change analyses applied to data, create new reports or change existing ones, and so on.

In [45], SSBI is presented as an important promise of BI, *despite the current difficulties in making those softwares easy to use for business people*. SSBI has been the center of some attention from specialized institutions (e.g. TDWI [78], Gartner [123] or Forrester [50]), which is another clue that business users are actually interested in achieving shorter time-to-value and doing BI on their own.

8.3 Why Make Self-Configurable Systems?

SCS are ways to deal with the change of requirements. Instead of making a system that satisfies exactly all of the most specific requirements identified at design-time, we consider these merely as examples of requirements that may arise at run-time, and we engineer features whose combinations can satisfy these anticipated, and perhaps some of the unanticipated run-time requirements.

In fact, the problem with design-time requirements is not that they are changing. In practice, many RE approaches exist, that can be used to identify new requirements [159] or track evolution of existing ones [161, 124]. The translation of those requirements into specifications for the system-to-be is not a problem either, as it has also been discussed at length in RE [145]. *The problem is rather on the time it takes to go from there being a new run-time requirement, to the time when the system has been changed to satisfy it.*

In the case of BI systems, the IT department is expected to be very responsive, and to provide quickly adapted solutions to business requirements. This is a generic requirement from BI systems, also known as time-to-value: once a BI sys-

tem is implemented, users must gain easy and rapid access to information, so that the decision making process remains efficient [78]. Traditional RE approaches can appear limited in that regard, because they assume elicitation, modeling, analysis, verification, negotiation, validation, and so on, have to be done for new requirements. This can influence time-to-value negatively.

As a response to delays due to *changing requirements*, SCS attempt to transfer some of the design responsibility to end-users, who are therefore in charge of understanding what they need, and directly designing what is required to satisfy these needs using a SCS. For example, in BI, Self-Configurable Business Intelligence (SSBI) is used to enable end-users to select some data sources and decide about a visualization tool to view it, all by themselves. Ultimately, they choose what to include in the report that they need the system to make.

Unlike for classical BI systems, SSBI does not require the usual RE processes to occur each time a user has a new requirement. There is therefore a tendency to make systems with generic features, and expect users to combine / configure the latter by themselves, in order to satisfy their new, specific requirements on-demand. They do so with the support of the system, but without the intervention of system engineers. This approach has the main advantage of reducing time-to-value, and hence improving users experience of the BI platform [78].

8.4 Indirect Requirements Satisfaction

In this section, we suggest and discuss an essential property held by typical requirements of SCS. We consider that any requirement which satisfies this property can be characterized as a *Self requirement*. The more a system is specified by such SCS requirements, the more the system can be considered as a Self.

It implies that the distinction between SCS and non SCS is not clear cut, and that any system can be placed on a "Self dimension". Besides, SCS can be distributed or centralized, adaptive or not, agent-based or otherwise, and so on - all such system categorizations are orthogonal to the Self dimension.

To introduce the property in question, it is important to distinguish users from other stakeholders of a Self. The reason lies in the difference between how a Self can satisfy user requirements, and how it can satisfy those of other stakeholders. Just as any system, a Self may satisfy the requirement of a stakeholder who is not a user, e.g., a stakeholder may not be involved in using the system, but may require that the license to use the Self costs below some amount per year. If the annual license costs X, and is smaller than Y (the maximal amount that this stakeholder set), then the system satisfies the requirement. We say that the requirement is *directly* satisfied, *since this stakeholder does not need to invest any more effort in using the system, in order to satisfy that requirement*.

The story is different for most users of SCS, i.e., individuals who will interact with the system at run-time, precisely in order to satisfy their own requirements. If a system directly satisfies the requirements of its users, then it does not belong to SCS. It does belong to SCS, if it satisfies these requirements *indirectly*. We say that a system satisfies a requirement indirectly, if there exists a scenario for using that system, such that if the system is so used, then the requirement will be satisfied, *but that is not the only possible scenario for using that system, and that scenario is not necessarily known by either the users or the system designers.*

It looks like quite a lot of software only indirectly satisfies their users' requirements. An operating system satisfies indirectly a user's requirement to print out a document, if the user needs to find, install, and configure by herself a printer driver. A word processor indirectly satisfies the requirement to format a text according to some formatting guidelines, because it is the user who has to figure out how to use headers, footers, front pages, blank pages, and so on, in order to ensure that the document does indeed follow the guidelines. A web browser, in contrast, looks to be directly satisfying its main requirement, which is to display content on the World Wide Web. But it indirectly satisfies the requirement to play specific kind of video files, if the user has first to find, download and install the relevant plugin.The usual calendar applications satisfy directly the requirements to add events and reminders, invite people to join events, and such. But they only indirectly satisfy the requirement to find the slots that suit everyone, when organizing a meeting. To satisfy that requirement, the user has to find some clever way to use the various existing features in her calendar application.

It is an essential property of SCS that *the intention in designing them is to satisfy many user requirements indirectly*. The consequence is that a user will have to do the work of finding the appropriate scenario that mobilizes the features of the system, in a way which will satisfy this user's requirements. The scenario should not already built into the system in such a way that a user can, without much effort, activate it. If the intention in designing a system is primarily to enable the indirect satisfaction of many requirements, which were anticipated or unanticipated at design-time, we will say that the system is *undetermined*. We will say that the system is *determined*, if the intention in designing it is to satisfy exactly some specific set of requirements identified at design-time.

8.5 Requirements from SCS and non-SCS

In order to analyze the way RE happens in the particular case of SCS, we first suggest to distinguish between two types of requirements. Then, using this distinction, we provide a more accurate, RE-oriented, definition of SCS.

	Determined	Undertermined
Business Users	Google Calendar, Microsoft Outlook, Microsoft Word, Skype	Blogger, Joomla, Matlab, QlikView, Excel PowerPivot/View
IT Experts	Visual Paradigm, FileZilla, AVG Anti-Virus, Apache Web Server	MicroStrategy, Pentaho, Symfony Framework, .NET Framework, Java

Table 8.1: Examples of Software

We start by distinguishing two kinds of requirements, called Stakeholder Requirements (R_{Stk}) and Derived Stakeholder Requirements (R_{DS}). As their name suggests, we obtain R_{Stk} from stakeholders and through requirements elicitation, which may involve interviews, observation, documentation analysis, and so on. R_{DS} are the requirements that a requirements engineer defines herself, on the basis of R_{Stk}. R_{DS} are made, for example, by refining, decomposing, disambiguating, or otherwise manipulating R_{Stk}, in the aim of identifying such R_{DS} which are operational. A requirement is operational when there is a specification which can be implemented (that is, its implementation is judged feasible), and there are good reasons to believe that, if implemented according to that specification, the resulting system will satisfy that R_{DS}. Various frameworks exist to identify variability in goal models, and could be of particular interest in the identification of R_{DS} [65, 99]. We will say that R_{Stk} and R_{DS} together are *Ground Requirements* (R^G), i.e, R^G is the union of R_{Stk} and R_{DS}.

We then distinguish R^G from *Self Requirements* (R^{Self}), which are defined by requirements engineers, in order to ensure that the system has generic features, whose various combinations could satisfy potentially many R^G. For example, the user of a BI system can have the requirement r_1 to "Display average sales margin per product". Such requirement has been elicited explicitly from that business user (through, for example, an interview): r_1 is therefore part of R_{Stk}, and hence of R^G. That requirement provides some direction to derive requirement variants. For instance, r_2, "Display sales revenue repartition per vendors", can be derived from r_1 and is then part of R_{DS}, hence also of R^G.

SCS are, however, not made to satisfy specifically R^G. Instead, they are made to satisfy a requirement r_3, which is "Be able to *[displaying]* an *[arithmetic function]* over one *[business fact]* for one *[business dimension]*". The latter requirement is such that, if it is satisfied, then both r_1 and r_2 will be, but also potentially many others similar to r_1 and r_2. The requirement r_3 is in R^{Self}. The difference between r_1 and r_3, and between r_2 and r_3, is that r_3 is obtained by looking at r_1 and r_2, and finding what is common to them, in order to formulate a new requirement which, if satisfied, would lead us to conclude that both r_1 and r_2 are satisfiable, provided

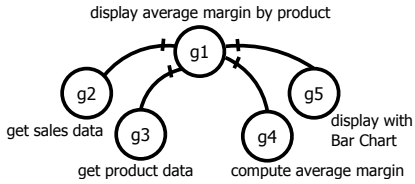


Figure 8.1: Goal refinement of r_1

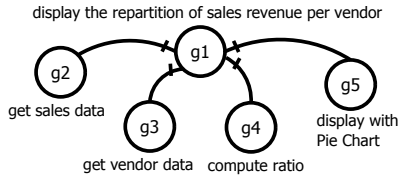


Figure 8.2: Goal refinement of r_2

that the users find out how by themselves.

8.6 Requirement Engineering for SCS

SCS vs Non-Self: An Illustration

Previous RE definition of SCS offers a support for distinguishing between the design concerns of SCS and non-SCS. Consider previous example r_1, where a user wants to "Display average sales margin by product". To obtain such result, the user can use a BI solution, and has two alternatives, called A and B below.

Alternative A: she could use a classical BI system. In that case, she would have to ask the IT to design a report, which shows the average margin by product. This results in the stakeholder requirement r_1 which is part of R^G. One could model that requirement via a goal model such as in Figure 8.6. With R^G, the IT could decide about the design of a new report, with no room for Self-Configurable: user might simply need to select a product to obtain the information she needs. Here, there are no features to select: everything is decided for the user in advance, so that the system can be said to be Determined.

As discussed in our Introduction section, the requirements from a BI solution are likely to change rapidly. Previous users could for instance have the new requirement r_2, illustrated with a goal model in Figure 8.6. To be achieved, that goal would require a new round of elicitation and operationalization, as r_2 would be added to R^G. Based on that new R^G, IT would have to design a new report. This repetitive RE process can increase time-to-value.

Alternative B: The user has access to an SSBI solution, and she is responsible for satisfying her requirements. Suppose that this is a simple spreadsheet software, such as Excel. Let there be a spreadsheet, which satisfies r_3 mentioned earlier. Starting with her first requirement r_1 (Figure 8.6), the stakeholder could for example select some rows from a data set she judged relevant, sum the cells and divide the result by the count of rows. She could also select all the data, apply a filter to it to keep only the last six months, and compute the average using the function for computing the mean, and so on. This system is Undetermined, as it is up to the stakeholder to find and design a solution to her problem. If a new requirement arises, let's say r_2, there is no need to re-engineer the SSBI solution.

The user, or someone helping her, would simply adapt some part of her initial solution to design a new solution that satisfies the new requirement.

A RE process adapted for SCS

From an RE perspective, Alternative A and B imply different design approaches. This is illustrated in Figure 8.6. In Alternative A, engineers have to decide about a specification that satisfies the Ground Requirements they elicit from business users. Only R^G is used to design the system-to-be. If R^G changes (due for example to a new variant of a requirement), then engineers must redesign the existing software to satisfy that new set of requirements. In Alternative B, engineers must identify user requirements, and then try to anticipate any other possible requirement. This results in a set of generic requirements R^{Self}. The design based on R^{Self} must offer sufficient features for the user to satisfy by herself the requirements that may appear at run-time in R^G.

Actually, the design of a Self cannot work on R^G since operationalizing R^G would consist in delivering a determined system providing business users with all required features in a single design. It is illustrated in the goal model GV in Figure 8.6. Identifying SCS Requirements is more than only taking into account all possible variability in users requirements. Although possible Self configuration would be able to eventually operationalize each leaf node of GV, a system which directly operationalize all leaf nodes of GV is not a Self. From there on, non-Self systems build from the operationalization of requirements in R^G, while Self systems build from the operationalization of requirements in R^{Self}.

Nonetheless, setting up R^{Self} from R^G is currently still a research challenge.

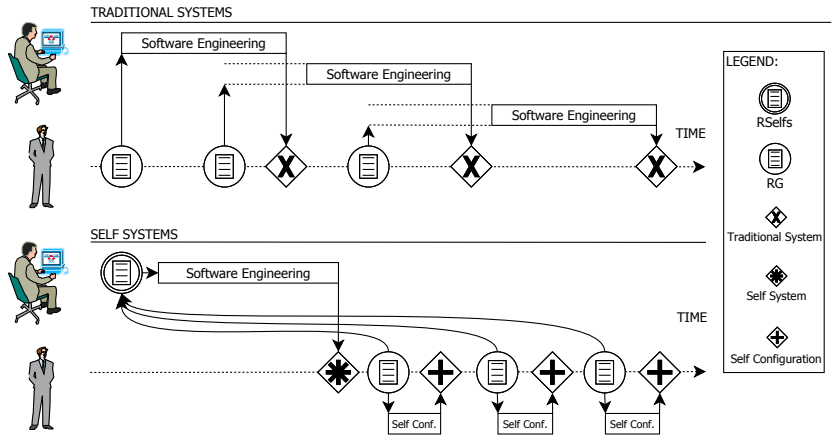


Figure 8.3: Comparison of RE process for traditional and Self systems

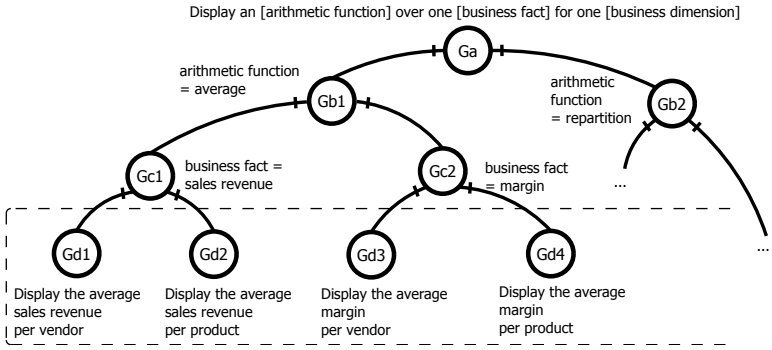


Figure 8.4: The goal model GV

Current methodologies only focus on R^G, i.e. how to gather and model R_{Stk} as well as how to derive R_{SD}. To the best of our knowledge, little attention has been paid as to how R^{Self} can be *abstracted* from R^G.

Notice that Self requirements open the way to some unanticipated uses of the system. Consider the case of MS Word. Word proposes to its user a mailing functionality, in which users are capable of selecting themselves fields, displaying the latter on a form, defining the layout for these fields, etc. Word is therefore somewhere on the Self dimension between pure SCS and pure non-SCS, because it satisfies at least one Self requirement, i.e., the user is able to define a mail on her own. A side effect of being a Self system is that users may be creative in using the software, e.g., defining a mailing is not the only way to use the Word functionality. In practice, it could be used in some other, unanticipated, ways.

Imagine for example a professor who wants to create several exercises for her students. Each exercise will be the same except some values and some words which will be changed. For this purpose, she could use the mailing functionality to design a template with print labels, and generate several exercises by giving different values for each label. In this context, she used an Undetermined functionality of Word in order to design her *own solution*. The use of a Self therefore depends on the creativity of its users, and should not be limited to the use cases for which it was initially designed.

8.7 Challenges of SCS for Requirements Engineering

Although critical to ensure a Self anticipates as much as feasible of future stakeholders' requirements, the identification of R^{Self} based on R^G (using, for example, an abstraction mechanism) presents some risks. Using the full set of R^{Self} to decide about the specification of a Self can lead to systems with numerous features, and hence to relatively complex SCS. By complex, we mean that they provide

many features to end-users. The problem with the complexity of a Self is that it can be negatively correlated with its usability: end-users, who have relatively low (sometimes no) IT background, may not be capable of understanding and combining the features consistently to build custom solutions. That problem of complexity is typical of SCS: regular systems avoid such complexity by directly satisfying the specific R^G.

Consider again the requirement of a user who wants to "Display average margin per product". Imagine the user has to satisfy that requirement using a Self. She is given a spreadsheet software such as Microsoft Excel. Excel contains Self requirements because the user is in charge of designing its own solution to compute her average margin from a range of data. In Microsoft Excel, she may have the choice between five, maybe six, features (average, count, sum functions, etc.) to be combined in order to compute an average margin. Imagine the same user is exposed to a new, more complex system, with hundreds of features that could be used to compute that same result. That system would be less usable for the business user. There would be risks that the user gets lost, or uses inappropriately some features of the software, with the ultimate risk that this business user does not satisfy properly her requirements.

This threat is important, and is reflected in existing SSBI solutions, where users are often discouraged because the Self is too complex for them. For example, Weber emphasized that "In an effort to give users what they want, IT sometimes errs on the side of giving users everything" which he claims is a typical problem of SSBI system [151]. SSBI experts also highlighted that "It turns out that most users found the tools too difficult to use. Even when the tools migrated from Windows to the Web, simplifying user interfaces and easing installation and maintenance burdens, it was not enough to transform BI tools from specialty software for power users to general-purpose analytical tools for everyone in the organization." [45]. In that regard, we consider there is a gap in current RE approach to SCS: designers should not only be interested in creating systems that satisfy the set of requirements R^{Self} (such as in SSBI). They should also account for the fact that SCS must be usable for business users. Therefore, they should pay attention to the number of feature they provide.

Note finally that research has been conducted to bring variability into software development. One of the most important research regarding variability is Software Product Line Engineering [117]. Although it aims to build a base system which can be customized to particular needs, this customization still requires IT intervention. Moreover, it does not aim to transfer the design responsibility of the users. Consequently, RE is traditionally about R^G and how to derive products which implement sub-parts of R^G. To the best of our knowledge, no research has gone on the business-user intervention in the resolution of R^G.

Now that we have discussed theoretical background of SCS, we will present how to model the SCSRP in Techne and then present a mathematical problem able to resolve the trade-off problem. The approach is illustrated with the MystShop case.

8.8 MystShop Case Study

The interesting part of the MystShop Case for this chapter is related to the period when potential new customers appear and the business wanted to create examples of checklists for those prospects. Up until this time, the report was filled in a webform hard coded for a single customer. With those new customers, custom checklists were required. We will illustrate the example with two new customers called hereafter them H&N and ekoa. Since each customer has its own needs regarding mystery visits, that is, uses different sets of criteria, MystShop will have to introduce new forms in the systems, one per customer. The question is the following: should MystShop IT have the responsibility of developing each new form or should this responsibility be transferred to users which could configure themselves each new form? There are various benefits to the second approach, such as faster adding of new forms in the system since users can do it themselves. Secondly, it allows IT to focus on developing features with higher value-added. However, this solution adds complexity in engineering and development, and makes it harder for users who need to know how to add the new forms.

8.9 Modeling

In this section, we discuss how the Requirement Problem of Self-Configurable Systems can be modeled. The MystShop case is used to illustrate our discussion. The Requirement Modeling Language (RML) used is Techne [85]. We use Techne because as it is based on a generic ontology of requirements and its reasoning rules can be mapped to an optimization problem [57].

The MystShop case will be modeled as follows. First, we discuss the Requirements Problem (RP) when there is only one customer. Then, we show that when considering possible evolution of the system for several customers, it becomes preferable to introduce a syntactic extension in Techne. This extension is aimed at modeling a specific requirements pattern. Finally, we suggest introducing a *Resource* concept to Techne in order be able to model consumption of resources when executing tasks. Resources can be work hours of different actors, raw materials or any other consumables. In the MystShop, we will distinguish consumption of development hours and business people efforts. This is key information for deciding what should be self-configurable in the system.

All modeling aspects discussed hereafter are supported by an online tool called AnalyticGraph. The tool is freely accessible, and goal models presented in this chapter are also accessible to any reader via the given URLs.

Basic Goal Model for MystShop

The main goal of MystShop is to ensure that *"Results of visits at VM stores are filled in custom forms"*, which has the identifier G0. After refining this goal, we can find that it would be satisfied if mystery shoppers do the task *"Perform the visits at VM stores"*, T3, and if goal *"Custom form for VM is designed"*, G6, is already satisfied. These two premises are linked to G0 via an inference node I2. This inference means that iff premises are satisfied, then the conclusion node can be considered as satisfied as well. This model is shown in Fig. 8.5.

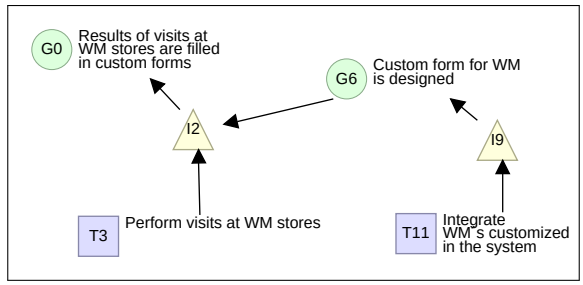


Figure 8.5: Goal Model of MystShop Requirement Problem (accessible at `http://www.analyticgraph.com/app/?g=0QBsZgJvsH`). Goals are represented as circles, tasks as squares and inferences as triangles.

As we said earlier, MystShop does not have only one customer. In its case, there are three customers: WM, H&N and ekoa. Although customers are different, the requirement problem and consequently its modeling remain similar. Instead of filling the form for WM, mystery shoppers will have to fill a customized form for respectively H&N and ekoa.

This is a situation where an SCS becomes a possible approach. Requirements can evolve over time: new customers can arrive, current customer will frequently change their criteria, they could ask new scale levels, demand attached pictures and so on. In these requirements change, we can identify new requirements such as to attach images to the visit report. There are other requirements which imply less development such as modifying customers' form. For those new requirements, it seems more interesting to delegate some design tasks to user (configure new forms) while developers would be focused on entirely new features (such as having the possibility to notify the customer with an email when a visit was done).

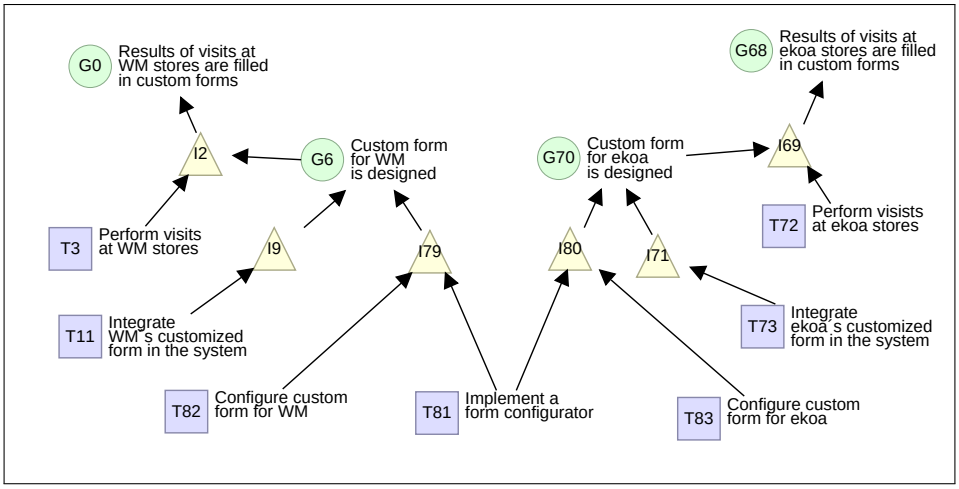


Figure 8.6: MystShop Goal Model for two customers (accessible at `http://www.analyticgraph.com/app/?g=UeLvE0nU51`)

Modifications in the goal model are the following: We replicate for new customers what was done for WM. Then, we consider the possibility to implement a form configurator, which would allow users to configure new forms when needed. This model, for WM and ekoa, is in Fig. 8.6. Notice that T81 *"Implement a form configurator"* can contribute to design forms for both VW and ekoa. From a modeling point of view, it could be interesting to have a generic notation to more easily represent these requirements patterns.

Introduction of Variation Points

In order to introduce the three new customers without replicating the model three times, we suggest a simple new notation. This new notation does not by itself extend Techne. It is just syntactic sugar used to refactor models which have specific patterns.

For example, when goal model in Fig. 8.6 is analyzed, it is easy to see that its left part is similar to its right part. The only difference is the name of the store (either VM or ekoa).

We suggest that the pattern could be factorized, which gives the model in Fig. 8.7. All similar nodes are merged and a variation point (V391) which enumerates a set of variants (here VM, H&M and ekoa) is linked to them. Nodes which depend on the variation point have a dashed border. This notation becomes truly interesting as the number of variants increases.

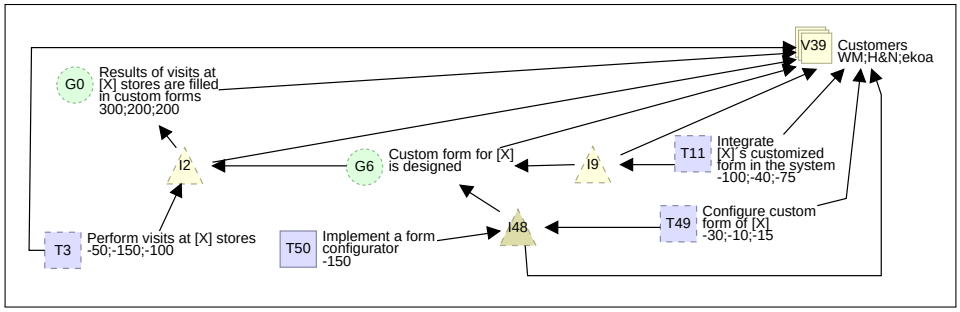


Figure 8.7: MystShop Requirement Problem with Variation Point.

But as previously discussed in the introduction of this chapter, one of the main problems of SCS is finding the right balance between self-requirements and other requirements. As the key difference between SCF and non-SCF is who (users or engineers) is responsible for satisfying them, we need to be able to specify who is responsible for each task.

Introduction of Resource Nodes

We decided to add a new *Resource* concept to Techne. The corresponding node (depicted with a stick-man symbol) is used to represent instances of this concept. A resource in our model can be a human or a role that will perform a task (the reason why we chose the stick-man symbol). In that sense, it is to some extend similar to Actor entities in i^* as resource are task performers and they will depends on each other to achieve goals [154].

However, it differs from i^* actor as it is less focused on the intentions carried by the concept[4]. While i^* stresses the intentions that each actor has and the mutual dependencies between actors, a resource in our framework is mainly considered as a task performer. If a task is linked to a resource, it means that this task requires work from this resource to be executed. For each resource, a budget label must then be specified, which determines how many units of this resource are available to perform all the linked tasks.

An example is provided in Fig. 8.8. It says that tasks T3 and T49 must be realized by Mystery Shoppers while T50 and T60 must be realized by developers.

[4]It could be interesting to reconcile both concepts (i.e. i^* actor and our resource concept) which would open possibilities to introduce multiple utility functions. As discussed in Section 4.2, single utility function is a limit of our framework but potential solutions based on the *actor/resource* concept are sketched in the further work section (see Section 10.2).

For convenience, the variation point node is hidden from the diagram but we can still notice variable nodes thanks to the dashed-border they have.

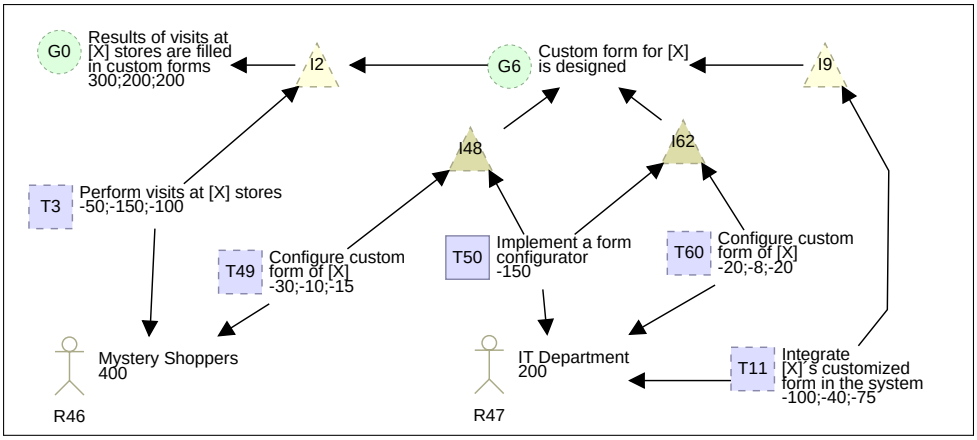


Figure 8.8: MystShop Requirement Problem with Resource Nodes. The Variation Point is Hidden for Visibility Purposes. (This model is accessible at `http://www.analyticgraph.com/app/?g=XnTuTK9p8g`).

AnalyticGraph.com as Supporting Tool

All previously presented models and modeling primitives are accessible on AnalyticGraph. This is a web-based platform designed to support modeling and reasoning on RE models. In AnalyticGraph.com , users model RE problems as directed graph. Each graph consists of a set of nodes linked by directed relations. The graph is stored in a graph-oriented database and meta-data (e.g. graph name, author...) are stored in a relational database. Currently, AnalyticGraph.com comes only with Techne as RML, but other RMLs can be specified and used.

The case presented in Fig. 8.8 can be accessed at `http://www.analyticgraph.com/app/?g=XnTuTK9p8g`. New modeling primitives (Variation Point node and Resource node) are accessible in the Self-configuration notation on the platform. Tutorials are available[5]. Among them is a tutorial related to the SCS dedicated module. This module is aimed at transforming the goal model in a Mixed-Integer Program (MIP). The language used is the GNU MathProg language [6]. The mapping between the goal model and the MIP is described in the next section.

[5] `http://analyticgraph.com/category/tutorial/`

[6] `https://en.wikibooks.org/wiki/GLPK/GMPL_(MathProg)`

8.10 Problem Formulation

In this section, we present the mapping between the goal model and a Mixed-Integer Program in the specific RE issue of self-configurable systems. The MIP formalizes the problem of finding the mix of Self-Configurable Features and other features. We first show how resources nodes are introduced in the MIP and then discuss the consequences of having variation points. The formal definition of the mixed-integer mathematical model is in Table 8.1.

The *objective function* of this optimization problem is the maximization of the value of satisfied goals (including softgoals and quality constraints) and selected tasks. That is the product between the node satisfaction and its value.

$$\max \sum_{n \in N} \mathsf{u}(n) * \sigma_n$$

MIP 8.1. Description of the basic MIP for a SCS Goal Model in Techne.

Sets		
N	$= G \cup T \cup S \cup Q \cup D$	Concept nodes
M	$\subseteq N$	Mandatory nodes
C	$= \{c_0, \ldots, c_i\}$	Conflict nodes
I	$= \{i_0, \ldots, i_i\}$	Inference nodes
R	$= \{r_1, \ldots, r_i\}$	Resources
Decision Variables		
σ_N	$= \{\sigma_i \in \mathbb{B} : i \in N \cup I\}$	Binary variables representing satisfaction of nodes.
Functions		
u	$: N \rightarrow \mathbb{R}$	Utility function
B	$: R \rightarrow \mathbb{R}$	Resource budget
$\mathsf{in}(x)$	$\subseteq N \cup I$	Incoming nodes function
$\mathsf{in}_Y(x)$	$= \mathsf{in}(x) \cap Y$	Typed incoming nodes function
Objective function		
	$\max \sum_{x \in N^*} \mathsf{u}(x) * \sigma_x$	
Constraints		
Premises constraints $\forall i \in I$:		$\sigma_i \leq \sum_{x \in \mathsf{in}(i)} \frac{\sigma_x}{\lvert\mathsf{in}(i)\rvert}$
Conclusion constraints $\forall n \in \{x \in N : \lvert\mathsf{in}_I(x)\rvert > 0\}$:		$\sigma_n \leq \sum_{i \in \mathsf{in}_I(n)} \sigma_i$
Conflict constraints $\forall c \in C$:		$\sum_{x \in \mathsf{in}(c)} \sigma_x \leq 1$
Mandatory constraints $\forall i \in M$:		$\sigma_i = 1$
Budget constraints $\forall r \in R$:		$\mathsf{B}(r) \geq -\sum_{t \in \mathsf{in}_T(r)} u(t) * \sigma_t$

Budget Constraint

In order to know what is the best allocation of tasks, we need to introduce a budget constraint. Indeed, without such constraint, the best allocation would be to give all tasks to the most efficient resource. However, resources have budget constraints (in monetary value or in time). As illustrated in our introductory example (Section 8.1), workers work 40 hours a week.

Then, we can add such consideration in the MIP by adding a new set R representing all resources considered in the goal model. Then, a new budget function denoted B (defined on R) relates each resource to its budget limit. Finally, a new set of constraints called *Budget Constraints* is added to the MIP. It states that the utility value (i.e. cost) of all satisfied tasks a resource is linked to, should not exceed its budget limit.

Variability Aspects

We defined four possible ways to use the variation point, as introduced in Section 8.9. These cases are depicted in Table 8.2. The mapping with the MIP is also provided. The presented rules have to be applied from sink nodes to source nodes.

Moreover, since a variable node (i.e. a node linked to a variation point) actually represents different variant nodes (i.e. one for each variant of the variation point), utility attached to this node is no more scalar. For example, consider our previous MystComp example: "Perform visits at VM store" should cost 50 while "Integrate VM's customized form in the system" cost 100. If we consider another MystComp's customer, values can be totally different. Consequently, utility attached to this node is a vector of the same size as the number of variants in the variation point.

8.11 Resolution

The solution provided by the optimization model consists for each variant of the set of tasks to undertake in order to maximize expected utility. Since tasks are related to resource, we also know who should be responsible of each tasks. As an example, consider Table 8.3 which depicts satisfaction values and utility values.

8.12 Limitations

There are several limitations to point out.

First, while we can expect that configuration costs would increase exponentially as features are added, cumulative configuration costs supported by users

Table 8.2: Illustration of the four possible mappings of variation points

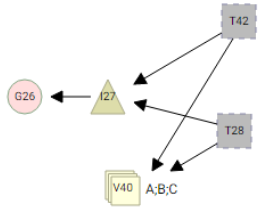
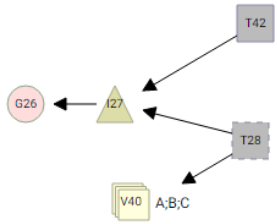
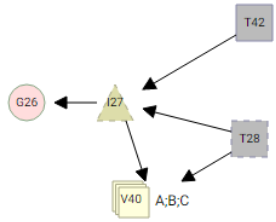
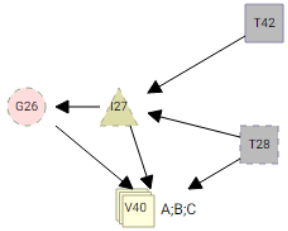
Premises are all variable	
	Conclusion Constraints: $G26 \leq I27$ Premises Constraints: $I27 \leq \frac{\sum_{x \in \{A,B,C\}} T28_x + T42_x}{6}$
Some premises are variable	
	Conclusion Constraints: $G26 \leq I27$ Premises Constraints: $I27 \leq \frac{T28_A + T28_B + T28_C + T42}{6}$
The inference and some premises are variable	
	Conclusion Constraints: $G26 \leq I27_A + I27_B + I27_C$ Premises Constraints: $I27_A \leq T28_A + T42$ $I27_B \leq T28_B + T42$ $I27_C \leq T28_C + T42$
The conclusion, the inference and some premises are variable	
	Conclusion Constraints: $G26_A \leq I27_A$ $G26_B \leq I27_B$ $G26_C \leq I27_C$ Premises Constraints: $I27_A \leq T28_A + T42$ $I27_B \leq T28_B + T42$ $I27_C \leq T28_C + T42$

Table 8.3: MystShop Case Solution

	T3	T49	T60	T11	G0	G6	Utility
VM	✓	✓			✓	✓	100
H&N	✓		✓		✓	✓	40
ekoa	✓	✓			✓	✓	85
					T50	✓	-120
						Total	105

Table 8.4: B&B Running Time

Nodes	Links	Time
50	50	0.05s
100	100	0.1s
100	150	1.5s
150	200	7s
200	250	100s

are polynomial in our model. However, one can get around this limitation by adding some additional nodes that would major cost of combined SCF.

Secondly, since we cannot know how many times users will have to configure the system (this depends on the frequency of requirement changes), configuration costs should be considered as stochastic variables.

Thirdly, this research discusses only the trade-off resolution of SCF and not their design challenges. This is outside the scope of the present thesis. The main way for our approach to reflect design challenges is via configuration cost.

Eventually, this model is a variation case of the *Knapsack problem* whose optimization problem is NP-hard. A dummy models generator has been developed to test large-scale graphs[7]. Some performance-related data are described in Tab. 8.4. The optimization problem is resolved with a branch-and-bound (B&B) algorithm but is has been proved that use of heuristics can improve performance in similar problems [7].

8.13 Related Work

Although there is increasing research on the Requirement Evolution Problem, we know only a few treatments of approaching requirements evolution with Self-Configurable Systems.

[7]The dummy model generator is accessible at `http://www.analyticgraph.com/dev/mod_opti/dummyConfigurator.html`

Tun et al. [141] used the Problem Frames Approach to create mappings between requirements and features, and between problem and solution structures, to support the evolution of a feature-rich software system. This helps manage of feature interactions as the software evolved.

Ernst et al. [48] posed the problem of finding desirable solutions as the requirements change and proposed to minimize the effort required to implement the new solution, which involves reusing parts of the old solution.

Silva Souza et al. [137] focused on Evolution Requirements (*EvoReqs*) whose are requirements that prescribe desired evolutions for other requirements that can be exploited at runtime by an adaptation framework.

8.14 Conclusion and Further Work

In this chapter, we provided in a first time an overall discussion about the use of Self-Configurable systems in organizations. We first discussed the rationale for such system, claiming that Selfs are valuable solutions to the problem of changing requirements and long time-to-value for business users. We defined Self-Configurable systems as being systems which contains operationalization of Self requirements. With such operationalization business users are in charge for configuring themselves the system in order to design their proper solution to some requirements. We then provided a deeper RE perspective on Selfs, by distinguishing between Ground Requirements, obtained requirements elicitation, and Self Requirements, which are requirements to be able to solve other, forthcoming, Ground Requirements. We then introduced on a discussion about the trade-off that may appears, during RE for Selfs, between the completeness of a Self platform (in terms of features) and the usability of the latter.

In a second time, this chapter suggested an approach to model and resolve instances of Requirements Evolution Problems which are addressed with Self-Configurable Features. The idea of such modular features is to give users a toolset, which they can configure in order to satisfy their future requirements. We explained that careful attention must be paid to find the right balance between self-configurable requirements and features, and others.

We suggested an approach to model this problem as a goal-oriented model. For supporting this modeling approach, we extended Techne with *Resource Nodes* and *Varation Points*. We then formalize the previously discuss trade-off problem as an optimization problem. Solution of this optimization model allocating tasks between users and engineers thus, determining the best mix. This second part is illustrated with the MystShop case whose artifacts are accessible on AnalyticGraph.com. Finally, we discussed limitations of the approach and carried out early testing to show how the implementation behaves with larger models.

Part III

Supporting Tool

9 Introducing AnalyticGraph.com

9.1 Introduction

The aim of this chapter is to present a tool that was developed in order to support previously presented methods. Beyond simply supporting an implementation of specific models presented, this tool allows users to develop themselves new models. For this, it allows to specify a new language (or extend an existing one) as well as an optimization model. However, the tool's main contribution goes further than the contribution to the Requirements Optimization Problem since it revives the old, but critical research topic of software tools for Graphical Requirements Modeling (GRM). We use the term GRM, to refer to the widespread practice in the RE field of representing requirements and their relationships in diagrams, and of doing reasoning (computation) on these representations. Typically, nodes in such diagrams hold information about the requirements, environment, or the system to build or change, while edges are labeled by relationships holding over the connected nodes; such relationships can say, for instance, that two requirements are in conflict, that some set of more concrete requirements refines a less concrete requirement, they may convey the relative desirability (preference) or importance (priority) of requirements, and so on. Various RMLs. [67, 44, 68, 107, 40, 154, 85] rely on GRM in the sense that models made with these languages are visualized as diagrams.

To this aim, we present and argue for a series of features that next generation GRM tools could have, and we present an early version of a new software tool, freely available at AnalyticGraph.com, which implements these features. The suggested features result from lacks we observed in other tools when trying to implement our approaches.

Our aim is not to propose a definite list of features for GRM tools, but instead to stimulate discussion and hopefully renew interest of colleagues in coming up and validating the relevance of new features for software tools which have received little attention and benefited from little innovation in the past years.

The chapter is structured as follows. Section 2 presents some existing GRM software tools and their respective features. Then, in Section 3, we discuss strengths

and weaknesses of existing tools. We introduce AnalyticGraph's architecture in Section 4. In Section 5, we show with examples how this architecture allows Analytic Graph to mitigate weaknesses. We finally conclude and discuss future work in Section 6.

9.2 Brief Tour of Existing GRM Software Tools

While vector drawing software can be used to do GRM, this is rarely, if ever advocated. It is common to see GRM being done with generic diagramming tools, which may or may not include graphical primitives corresponding to specific RMLs. We consider such tools as DIA, Microsoft VISIO, draw.io, yEd or OmniGraffle to be examples of generic GRM tools. Their limitation is that they offer little support to users in terms of syntax-checking and computation over the models (diagrams) made.

More comprehensive tools have been developed in RE to deal with such limitations. For instance, Tropos comes with the java-based GR-Tool, which proposes forward and backward reasoning on goal models [60, 61, 132].

In line with the Tropos methodology, TAOM4E supports a model-driven, agent oriented software development, and has been designed to respect the Model Driven Architecture (MDA) recommendations [11].

RE-Tools [138] is another modeling tool that supports notations such as i*, the NFR Framework, KAOS, Problem Frames, and UML. Among other things, RE-Tools supports the combination of previous notations, enabling engineers to combine functional and nonfunctional requirements, agents, goals, soft-goals, formal goals, and objects into one single diagram.

DesCARTES [94] is a Computer-Aided Software Engineering (CASE) Tool, which also supports i*, NFR models, UML models, and I-Tropos developments. It takes the form on a Eclipse IDE (Integrated Development Environment) plug-in.

jUCMNav [1] is another example of Eclipse plug-in which permits the modeling of requirements based on the User Requirements Notation.

MetaDONE [47] is a tool supporting domain specific modeling languages (DSML); it is aimed at helping engineers in the more effective implementation of software systems, based on the production of generative methods [92].

9.3 Strengths and Weaknesses of Existing Tools

Although existing GRM tools clearly differ in the type of language they support or the nature of reasoning they enable to perform on requirements models, they all seem to support requirements engineers in at least one of the following complementary ways:

- They offer symbols and visuals for a given notation, keep data and metadata related to these elements, and ensure these elements are combined in a way that complies with the syntax of the modeling language,

- They offer reasoning capability about a model to identify solutions, discover alternatives or resolve conflicts,

- They offer capability to design their own notation and behavior capabilities, fitted for the actual domain.

These are the three main pillars of GRMs, on which we continue to build Analytic Graph.

Beside those strengths, we see a number of improvements which could be made on existing GRMs, that would further help requirements engineers in the modeling and analysis of the requirements. Those improvements, which we discuss with more details in the remainder of the chapter, could be summarized as follows:

- *Portability*: models defined in one GRM, on one computer, are difficult to transfer to other GRMs/computers,

- *Collaboration*: existing GRM are not designed to ease collaboration between several engineers on same models,

- *Navigability*: models can get very large, and existing GRM offer no support to navigate effectively in the many nodes and edges that constitute a requirements model.

- *Reasoning*: current GRM offer predefined computations on models, leaving no room for custom reasoning,

- *Extensibility*: existing GRM are usually designed to support one or more preset RMLs, without the possibility for users to add new RMLs,

- *Flexibility*: it is normally not possible to make models made by bridging several models from different RMLs.

Our overall aim in making Analytic Graph is to consolidate the strengths and address the limitations of existing GRM tools. Although Analytic Graph is not fully developed, it is available for use and already illustrates many of its key features.

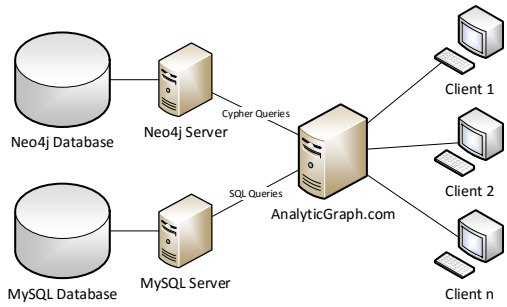


Figure 9.1: Architecture of AnalyticGraph.com

9.4 Analytic Graph Architecture

Existing GRMs are predominantly desktop tools, which hurts portability and collaboration. GRMs do not store and treat requirements models as graphs, which does not help navigability and custom reasoning. In this section, we describe how the architecture of Analytic Graph differs from classical GRM tools. Next section discusses how that architecture helps in relation to GRM limitations.

The architecture of Analytic Graph is illustrated in Figure 9.1. Our system is composed of one web-server on which AnalyticGraph.com is hosted. The web-server runs queries against a Neo4j server, which manages the storage of graphs. Neo4j is a graph-oriented and open-source database management system. It receives Cypher queries through a web-service from AnalyticGraph.com and returns json files with the result of these queries. AnalyticGraph.com also runs SQL queries against a MySQL server, to store or retrieve any meta-data on the graph (graphID, owner...) as well as data that is not related to a model; for example, requirements notations, logs, user account data, etc. Clients interact solely with Analytic Graph using a web browser, as explained in next sections of the chapter.

AnalyticGraph Stores Models as Directed Graphs

A limitation we emphasized in our introduction is that existing tools do not enable users to define and execute their own queries/reasoning algorithm on their models. To deal with this limitation, we designed Analytic Graph as a self-service system; we intend to let users define themselves the queries they want to run against a model, to define or import the requirements notations they want, with as little constraints as feasible on how the model is stored. The only constraints are those imposed by the RML which the user wishes to use.

This approach led to important constraints on the way RE models should be

stored in Analytic Graph. Namely, we need to store the model in a way that is amenable to flexible querying and computation, while at the same time reflect the structure of graphical requirements models. We therefore store models as graphs. Each concept and link of a model (for example, a goal, a task or a contribution link in i*) is stored as a node of a graph. Under this scheme, the directed links in our graph are simply used to connect two nodes of the model, and carry no other information than the direction. For example, a model in which a goal decomposes into two sub-goals will be stored in Analytic Graph as a graph composed of four nodes: one goal node, two sub-goal modes, and one decomposition node, with a link from each of the sub-goals to the decomposition node, and a link from the decomposition node to the goal.

The Techne model [85] already adopts such view on modeling. It represents a relationship between two nodes (goals, tasks, etc.) as other nodes (inferences, conflicts, preference), so that links (edges) themselves carry no information other than their direction. As already presented in previous chapter, Techne is already implemented in Analytic Graph and we use it in the rest of this chapter for illustration.

Analytic Graph is a Web Application

Another limitation we emphasized in our introduction is the difficulty for users to share and interact with models. We consequently built in collaborative features, namely to let users share the models they created in Analytic Graph, or to access and interact with others' models in few steps. We wanted a tool that does not require specialized software to be installed on users' devices, and we therefore opted for an on-line platform. Analytic Graph is a web application, in which users can create, edit, save and load models and RMLs. By associating a unique URL to each model and notation, it is also possible for users to share their own models, link models in documents, research papers, presentations, and elsewhere, and to access and/or import in their own library, the models or RMLs created by others.

9.5 Candidate Features for Next Generation Tools

In this section, we discuss features which we consider important for next generation GRM tools. We do this by presenting how Analytic Graph addresses common limitations of GRM tools. For each limitation, we present a feature of Analytic Graph, and we illustrate and explain how that feature works.

Portability

If a GRM tool is desktop software, then it needs to be installed on the users' devices, and models will be stored by default on these devices. *Our tool is a web application; it means that it is available on-demand anywhere, and requires no installation of specialized software*[1].

The main requirements for accessing Analytic Graph is to have access to an Internet connection, and have a web browser installed on the device[2]. Once logged-in, users are able to save their models. The models are stored on the Analytic Graph.com server, which means that the users can access the model on any device, using their credentials. Models are by default private, so that a model created on one user account will not be visible to other accounts, unless he gets the URL. We also leave the possibility for users to design models without having an account, in which case the model can be exported as an image or transferred to an existing account, or lost (not saved) otherwise.

Collaboration

With desktop GRM tools, a model is stored as a file and shared by sending the file among collaborators. If the model is shown in a research paper or other document, a reader needs to find the model source online, download and setup the relevant GRM tool, and only then work on the model, and use it in own research.

Analytic Graph was designed to simplify the process from seeing the model in a publication to being able to edit and run it. In the caption of Figure 9.2, we included a URL. Clicking on the URL should bring up the reader's web browser, and allow the reader to view, edit, and run on Analytic Graph the model shown in that Figure. It is a Techne model, of simple requirements for music streaming software.

Linking models for editing is made possible with two features of Analytic Graph: (i) *each model made with the tool gets its own permanent and unique URL*, so that models can be linked in research papers or other document types, and (ii) *any reader who has access to the URL of a model can click on the link, open a free account (or access as a visitor), and edit and run a copy of the linked model*.

Putting aside the graphical representation in Figure 9.2, any non-interactive visualization of the model – just as the one in that Figure and on any printed page and on any non-linked model – suffers from many drawbacks. For example, readers cannot run themselves computations on the model. They cannot reuse

[1] The tool is accessible at analyticgraph.com/app. Tutorials are provided at analyticgraph.com/category/tutorial/

[2] At this stage, Google Chrome is the recommended browser for accessing AnalyticGraph.com.

and enrich that model for their own purposes. They cannot navigate in the graph. And so on. Analytic Graph aims to facilitate the interaction with requirements models, by letting users access to and use the graph. As an example, we invite the reader to click the URL shown in the caption of Figure 9.2.

Navigability

We use the term navigability to refer to how one can search for and find sub-models of a given model. Sub-models may correspond to, for example, candidate solutions (specifications) to requirements, alternative refinements of a given requirement, all requirements which may be involved in a conflict, and so on.

Our approach to navigability in Analytic Graph is to *store requirement models in a graph database, use well known algorithms to search the graphs, and enable users to define their own queries on models / graphs*. A series of queries are predefined in Analytic Graph. Users open a model, select and click on the query to execute it against that model. Some queries are generic, in that they can be run on any graph, in an RML. Others are specific to an RML.

Queries we have been exploring so far do not add or remove nodes or edges in a model, but return a sub-graph. By combining several queries, users are then able to identify patterns in the model, select only a part of the model, identify a certain type of nodes, or identify the shortest path between two nodes. We see in this feature a first step toward more sophisticated queries and an interface for custom queries.

Consider our running example from Figure 9.2. The number of nodes is still limited, yet it might already be difficult to detect one particular set of nodes. Say for instance that you want to browse the graph in the refinement direction. You then need to locate sink goals in order to start the reading of the model. Using a simple search query, you could detect them instantly. The same search without a query would likely take more time, even if the number of goals is limited. Moreover there always is a risk you miss some of them. Similarly, one might run a query to count the number of conflicts, to select all soft-goals, to determine the entire sub-graph associated with one goal, etc.

Reasoning

Reasoning on a model allows engineers to answer questions, which the language was designed for. This is common to all RMLs, ranging from early requirement languages such as i* or Techne to later requirement languages such as BPMN, features diagrams and so on. For example, in feature diagrams, one question is to find some subset of interest, among all possible configurations of the modeled

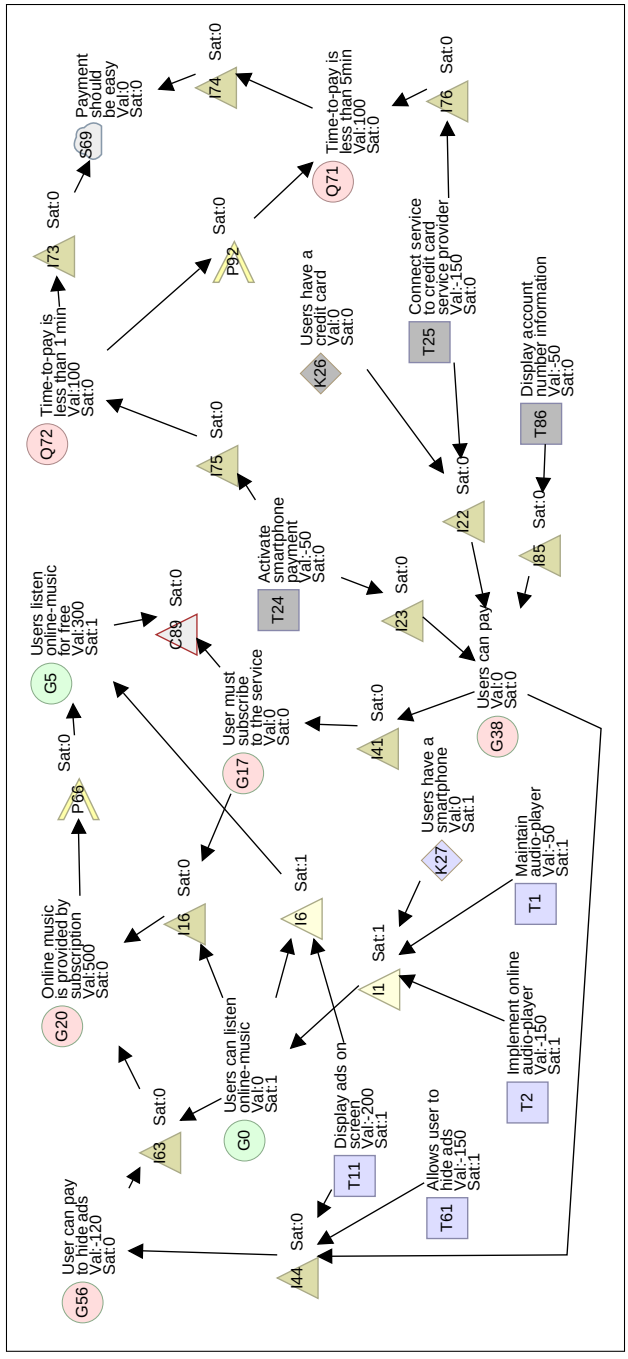


Figure 9.2: Example of a Techne Graph in Analytic Graph- analytic-graph.com/app/?g=5u3lNJVddv

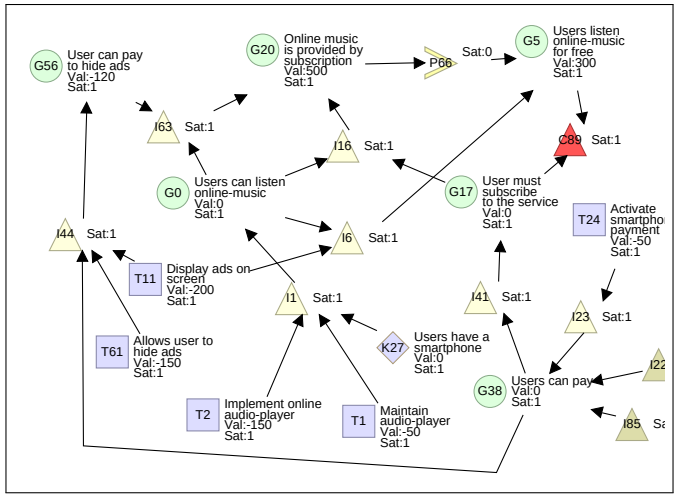


Figure 9.3: Example of Reasoning on Techne Graph in Analytic Graph- Illustration of the propagation of satisfying task T24

system. Without a proper tool capable of reasoning, the diagrams themselves are of limited use. Most GRM tools provide such features, but tend to be limited to one or more predefined reasoning ways on models.

Since Analytic Graph is based on a graph database, it comes with a manipulation language (called Cypher[3]) which has been used to design behavior on diagrams. Use of this language enables several ways of defining a behavior (transaction, node behavior), and leaves that decision to the user. For example, Techne mechanism of inference has been implemented by defining for each node a particular behavior. Each inference is associated with a query that evaluates if premises (i.e. incoming nodes) are satisfied, in which case it sets the conclusion node (i.e. the outgoing node) as true. This behavior is propagated from source nodes to sink nodes.

Consider our running example as illustrated in Figure 9.2. The number prefixed with **Sat** below the name of a node is its satisfaction; 0 means that the node is not satisfied, 1 means that the node is satisfied. Note the conditional formatting of nodes, depending on their satisfaction (another customizable feature). In our example in Figure 9.2, the goal G17 "*User must subscribe to the service*" is unsatisfied. During requirements analysis, engineers might want to test the effect on the realization of some tasks on this goal. Since satisfying goal G38 "Users can pay" should have an impact on G17, they just need to ensure the satisfaction of under-

[3]http://neo4j.com/developer/cypher-query-language/

lying tasks of G38. So, they set value of T24 "Activate smartphone payment" to 1 (true), and run the model again (we invite the reader to open the model and run node behavior in Analytic Graph app). The result is reported in Figure 9.3; the figure shows that satisfying T24 enables to satisfy the goal G38 of the model, yet introduces a conflict with G17 (symbolized by C89). It also satisfied among others G56, G0 and G20. This is a simple example of how Analytic Graph implements reasoning on goal models.

As discussed earlier, other reasoning or treatment of the graphs are possible. Consider now the case where utility values can be associated to each node in the model. It is represented by the numbers prefixed with **Val** next to each node. Positive value means a revenue while negative value represents a cost. Having this information, it is possible to map the model to a mixed-integer mathematical program and execute an optimization on this [57] (we invite the reader to open the model and run optimization in Analytic Graph app). Given these values, the optimal solution allows to reach a value of 350 with following node satisfied: G0 G17 G20 G38 Q72 K27 T2 T24 T1 I1 I41 I16 I23 I75.

Extensibility

GRM are often designed for a specific RML. For example, GR-Tool implements the Tropos notation, jUCMNav implements the User Requirements Notation, etc. Analytic Graph has a language management module, which allows users to define their own RMLs, both in terms of graphical primitives and their properties used for queries and computation. Users are also free to build requirements models using the notation they prefer. In case users need other notations, they can either design that notation or import it.

Although **Techne** is the only built-in RML at the moment, our aim is to increase the number of built-in RMLs available to users. The tool will soon allow users to define relatively simple notations of their own. They will do it using the *Language Management* (LM) tool[4], which enables users to define virtually any language, as long as its models can be represented as directed graphs. The meta-model behind language definition is depicted in Figure 9.4. A *Language* has a name and is composed of several *NodeType*. Each of them is described by a name, a shape (graphical representation) and a behavior (currently a cypher query). The user can define how and how many *NodeType* can be linked together. Moreover, each *NodeType* comes with a set of *PropertyType* which are characterised by a name, a type and a default value. Reasoning in a language is introduced by defining functions over values of node properties.

[4]More information about how the *Language Management* will work can be found at analyticgraph.com/create-your-own-language/

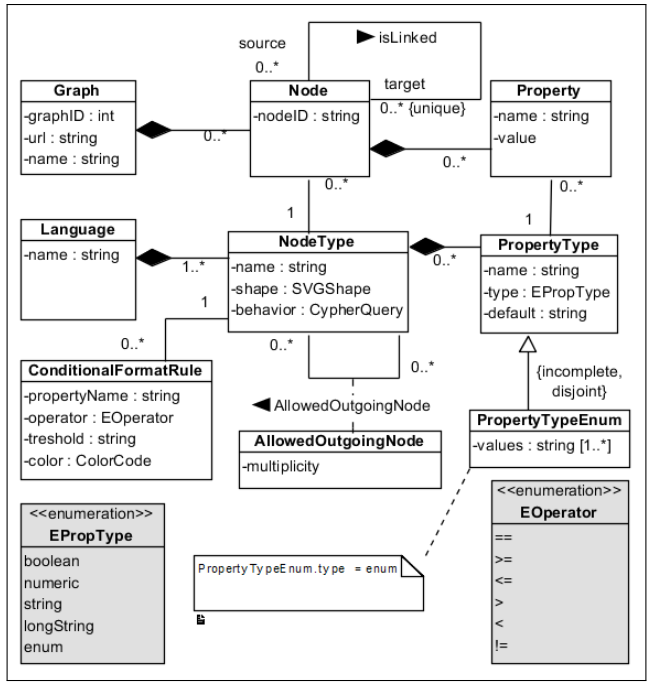


Figure 9.4: The Meta-Model of Analytic Graph Language Management Module

Flexibility

Different requirements notations have been defined to deal with different requirements engineering concerns. For example, Tropos [25] focuses on the agents from early requirements to specifications and implementation, i* [154] on intentions of stakeholders, BIM [77] on Business Intelligence aspects. It is usually not the case for a GRM tool to allow the building of combined models, that is, of graphs where sub-graphs are models in different languages, and where new types of relationships are used to bridge models of different languages. However, several approaches combining different notations have already been suggested. For example, Metzger et al. suggested to distinguish two types of variability by relating orthogonal variability models (OVMs) to feature diagrams [105]; two different languages.

We have made early advances in Analytic Graph to allow users to combine different languages in a same graph. Results coming from computing such sub-graphs are then used in other sub-graphs. For example, Analytic Graph comes with an Algebra notation capable of performing simple calculations such as addi-

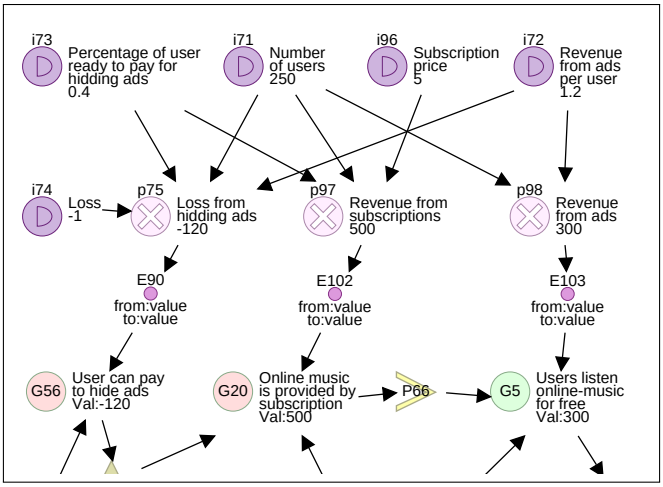


Figure 9.5: Example of Mixing 2 Notations in a Model - Application on G5, G20 and G56 from Example Presented in Fig. 2

tion, multiplication and so on. This notation has been used in the "Music streaming software case" in order to compute value of some goals. This application is depicted in Figure 9.5. Basic data such as the number of users, the price of subscription and so on, are entered by the user in order to compute value of goals G5, G20 and G56. These basic data are represented by nodes i71, i72, i73... The nodes are then used as inputs of multiplication (represented by cross nodes p75, p97 and p98) for computing advanced values (labeled next to the nodes). Results of the multiplications are next used as inputs of *Transfer nodes*. The latter are special nodes used to extract the value of one property of the incoming node and transfer it in a specified property of the outgoing node (the value of goals in this case). After running behavior of this graph, one can run the optimization model. This example shows how the Algebra notation can be used with Techne to perform a "what-if" analysis. Engineers will be able to check if the optimal solution is still the same if the expected number of users drops by 50%. In the future, users will be able to import data from external sources (a database, a web-service...)

Notice that the present discussion raises some important questions about the validity of using mixed RMLs; namely, does the combination of several different models – designed based on different assumptions and intended for different purposes – make sense? While this question is out of the scope of this research, it is worth noticing that part of our ongoing work goes into this both theoretical and empirical discussion.

9.6 Conclusion and Further Work - Toward Analytic Graph 2.0

Analytic Graph is still a prototype. This means that we are still working on the consolidation of our tool, to ensure it is fully functional and can be used properly by requirements engineers, as intended. Next steps includes the involvement of several users, as a way to collect feedback on Analytic Graph features, and improve/revise them, if relevant. Our future work will also investigate the feasibility and implementation of following improvements:

- Connecting Analytic Graph to external data sources such as databases, web-services,
- Providing classic diagram tool features such as undo, redo, auto diagram layout,
- Completing the user interface for defining custom languages, queries and algorithms,
- Improving real-time collaboration with features allowing users to work on the same model (and not only on a copy).

Despite all limitations of Analytic Graph today, we hope it illustrates interesting directions for the development of next generation GRM tools.

Part IV

Conclusion

10 Further Work

In this chapter, we discuss possible further research. Though, we think there are many directions that could be undertaken, we selected to discuss three of them. First, we suggest a next Requirements Issue to be studied as a ROP, namely the Requirements for Self-Adaptive Systems. Then, we show that it could be interesting to generalize the present framework to non-software related fields. Eventually, we discuss possible extensions to reduce the limitations induced by an aggregated utility function.

10.1 Requirements Optimization for Self-Adaptive Systems

One interesting additional Requirements Issue that could be tackled by the present framework is the requirements for self-adaptive systems. Those systems have the ability to adapt at run-time to changing user needs, system intrusions or faults, changing operational environment, and resource variability [42].

Actually, early work on the subject has already been undertaken. It resulted in the publication of :

> J. Gillain, S. Faulkner, I. J. Jureta, and M. Snoeck. Using goals and customizable services to improve adaptability of process-based service compositions. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–9. IEEE, 2013

The idea of this approach was to identify, at design time, sets and combinations of web-services able to satisfy a particular goal model. Some services being more fitted under particular conditions/assumptions. At run-time, when conditions of execution would have changed, a new configuration would have been suggested taken into considerations the satisfied domain assumptions.

As an example, consider to following example illustrated in Figure 10.1. The purpose of the system is to know the temperature of a certain geographical location in Celsius degree. In this case, since we know the latitude and longitude (K16 is blue), we have two solutions {F34} or {F36, F40}. We can then apply some

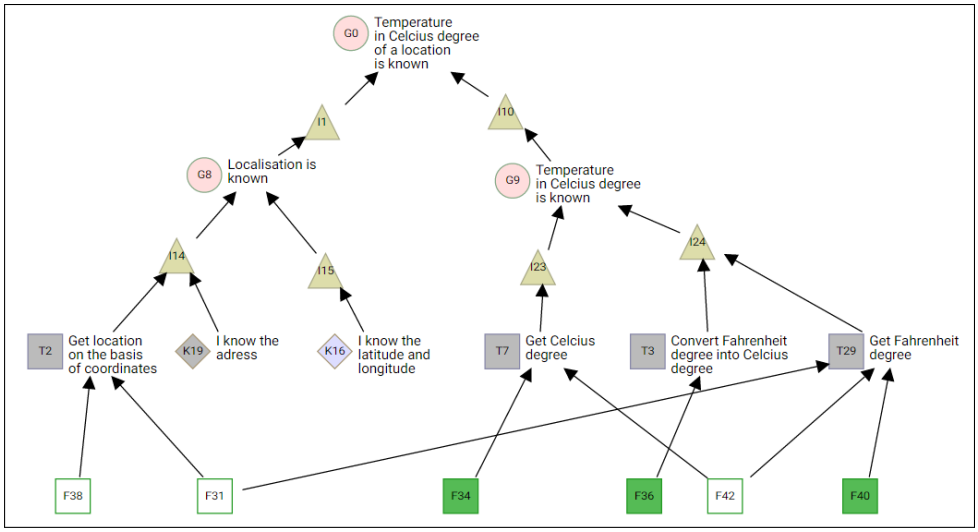


Figure 10.1: Goal model representing the initial situation of a self-adaptive system. Green squares are web services that can be used to execute the tasks they are linked to. Filled squares indicates available services while empty squares represents unavailable services. Diamond nodes are domain assumptions. If they can be considered as true, they blue filled, otherwise they are gray.

decision criteria to determine which one is the optimal (such as cost minimization). Then, if the assumptions change, such as in Figure 10.2 where the address is known instead of the coordinates and the availability of web-services has changed, we can re-run the optimization model. Then, a new solution, adapted to the new conditions would be either {F38, F36, F40} or {F31, F36}.

Nonetheless, a lot of research is still required on this topic because it simplifies too much the RP for self-adaptive systems since what really changes between requirements of such systems and the ZJ framework is that the former needs to specify additional requirements on how to go from one solution to another solution while the ZJ framework says nothing about this aspect [120]. Regarding this problem, it could be interesting to study if there exists several paths to go from an initial solution to a new targeted solution. If yes, we should study how we can represent those different paths and map them to MIP in order to determine the optimal path.

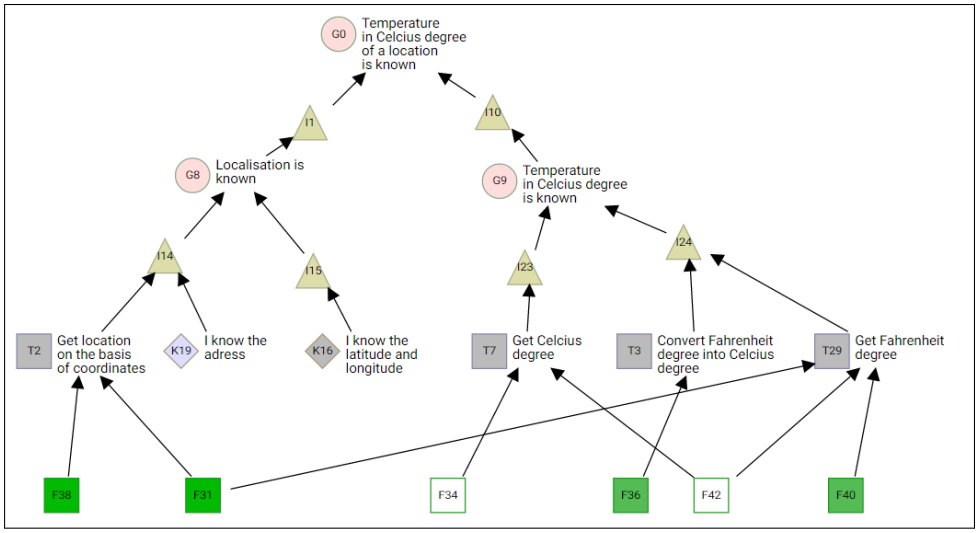


Figure 10.2: Modified situation of the goal model of for the example of a self-adaptive system.

10.2 Abstracting from Software Engineering

Another interesting extension of the current work could focus on its generalization. That is, apply it on other fields than Software Engineering. As examples of possible fields, let consider the building sector for the entire framework and the project management for AnalyticGraph. Those examples were selected arbitrary with the only purpose to demonstrate the potential generalization.

Building Sector: The entire present framework could be applied, for example, on a project of house renovation. Let consider that a house owner has several goals: diminish her energy consumption, reduce the humidity level and increase the available space. Each goal can be refined into sub-goals until some specific tasks. For instance, change windows, install an air cleaner system, insulate the walls and so on. We can compute the utility of each goal (e.g. estimation of the saved energy, the house owner willingness-to-pay for additional space...) and the cost of each task. We can also introduce a budget constraint. Then, we will have a model to optimize. Regarding this example, it seems that our framework does not need any change since this situation can be modeled with Techne.

Project Management: In another context, we could imagine using AnalyticGraph to model all the tasks of a specific project into a PERT diagram. Several

tasks being driven-effort, i.e. their duration can be reduced if we increase the number of resources working on it, there is some optimization of the resource pool to be done. The diagram would be enriched with the resources and their potential contribution on the different tasks (as we did in Chapter 5). Then, AnalyticGraph could compute the optimal allocation of the resource pool (with the hypothesis that everyone can work on everything). If the latter hypothesis does not hold, some additional constraints could be added to show that some resources can work only on some types of task. A tool such as MS Project[1] does not provide similar features (it only resolves over-allocation of resources by moving tasks). Here, the contribution would be on using graphical diagrams to specify optimization models instead of using linear programs. It would provide business users with more user-friendly tools for running optimization.

Goal modeling for multiple stakeholders

The objective functions of our optimization models do not distinguish stakeholders' utility functions. It means that they maximize an aggregated utility function without considering from who the intention originated. As discussed in the limitations (see Chapter 4), it can result in solutions where all goals of one particular stakeholder are satisfied while others have no goal satisfied.

We can however extend our framework by linking each goals to their respective stakeholders. This could be done by using the Resource node we introduced in Chapter 8. Then the mathematical model could require that a minimum level of utility for each actor should be satisfied. In that case, the Resource concept becomes similar to the concept of Actor used in i^* [154].

How to precisely model those considered constraints could be the subject of further research.

[1] https://products.office.com/en/project/project-and-portfolio-management-software

11 Summary

In this thesis, we suggested a new formulation of the Requirement Problem. We deviated from the framework suggested by Zave & Jackson, in order to incorporate an optimization dimension. The idea being to introduce capabilities to select a solution among several alternatives. Indeed, when focusing on requirements a system should satisfy, engineers face several potential solutions. Current formulations of the Requirement Problem lack of guidelines and criteria able to discriminate those solutions. However, it seems important to carefully select the most promising direction the system should take before deeper investigating this particular solution.

We decided to root this new formulation, that we called the *Requirement Optimization Problem* (ROP), into both the Core Ontology for Requirements Engineering (CORE) and the Mixed-Integer Programming. We demonstrated that several sub-problems of RE could relevantly be formulated with our framework.

More precisely, we focused and described in details four of them. All were illustrated with a single case: the MystShop case. This case, more used for illustrative purposes than validation, also helped us to identify and formulate concrete optimization problems (and their selection criteria).

Planning optimization

The first ROP we investigated was aimed at finding an optimal release planning. We rooted this problem into Agile methodologies in order to be more practicable. After briefly discussing the links between Agile concepts and goal modeling, we showed how to model such problems with Techne and how to map them to a MIP.

Reusability optimization

The second ROP was focused on finding the right balance of reusability between several systems to-be. This ROP was rooted in the Software Product Line Engineering context. We discussed the relevancy of optimization by demonstrating that there exists a trade-off between variability and commonality. We proposed a modelisation of the problem with Techne and map it to a MIP.

Cost minimization

The third ROP was aimed at linking our framework with existing formal cost estimation methods. Its objective was to minimize the cost of a system by comparing its different operationalizations. We selected the IFPUG Function Point Counting as the cost estimation method. We discussed the mapping between the counting rules and a specific MIP. We also had a discussion about the limitation this ROP suffers.

Self-configurability optimization

The final ROP was dedicated to find the right balance of configurability to be provided to end-users. We discussed why there is a trade-off when deciding if we should allow the user to configure himself a system. We introduced some extensions in Techne in order to be able to model such situation and we showed how to map the model to a MIP.

AnalyticGraph

Another major contribution of this thesis, was the development of a tool supporting the present framework: AnalyticGraph. It proved the feasibility of the discussed optimization models but it also suggested the capabilities next generation of RE software should have: portability, collaboration, navigability, reasoning, extensibility and flexibility.

Concluding Remarks

Aside from all the limitations the present work can have, it pursued the objective to acknowledge the importance on systematically consider and compare multiple solutions in Requirements Engineering. For the usual and inherent reasons linked to any thesis, we had to scope our work. We decided to focus on the association of CORE and MIPs. However, we think that many other perspectives could be taken to study Requirements Optimization Problems.

So, it is clear from our results that much research could be done on this subject. In any event, we hope to have initiated a new research direction that, we think, could lead to exciting further contributions to the field of RE.

Glossary

AnalyticGraph AnalyticGraph.com. i, 14, 16, 35, 63, 82, 115, 139, 149

ARP Agile Requirements Problem. 50

CAB Core Assets Base. 72

CLF Core Logical Files. 101

CORE Core Ontology for Requirements Engineering. 7

DET Data Element Types. 98

DF Data Function. 96

EI External Inputs. 103

EIF External Interface File. 98

EO External Outputs. 103

EQ External Inquiries. 103

ER Entity-Relationship. 92

FPA Function Point Analysis. 93

FPC Function Point Counting. 93, 97

FTR File Types Referenced. 103

GM Goal Modeling. 5

GRM Graphical Requirements Modeling. 149

IFPUG International Function Point Users Group. 93

ILF Internal Logical File. 98

IS Information System. 3, 12

MIP Mixed-Integer Program. 8, 28

NFR Non-functional Requirements. 5

NPV Net Present Value. 73

RE Requirements Engineering. 3

RET Record Element Types. 102

RI Requirements Issue. 24

RML Requirements Modeling Language. 7, 149

ROP Requirements Optimization Problem. 5

RP Requirements Problem. 4

SCF Self-Configurable Features. 124

SCS Self-Configurable Systems. 124

SPL Software Product Line. 67

SPLE Software Product Line Engineering. 67

SSBI Self-Service Business Intelligence. 126

TF Transactional Function. 96

UCP Use Case Points. 93

WACC Weighted Average Cost of Capital. 72

WTP Willingness-to-pay. 68, 119

ZJ Zave & Jackson framework. 4

Bibliography

[1] University of Ottawa: jUCMNav. http://softwareengineering.ca/ jucmnav/ (2011).

[2] *Function Point Counting Practices Manual*. International Function Point Users Group (IFPUG), 2000.

[3] *Quality Management Systems - Fundamentals and Vocabulary (ISO 9000:2005)*. International Organization for Standardization (ISO), Geneva, Switzerland, 2005.

[4] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 45–54, New York, NY, USA, 2012. ACM.

[5] M. A. Al-Hajri, A. A. A. Ghani, M. N. Sulaiman, and M. H. Selamat. Modification of standard function point complexity weights system. *Journal of Systems and Software*, 74(2):195–206, 2005.

[6] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6):639–648, 1983.

[7] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and software technology*, 43(14):883–890, 2001.

[8] Z. Bakalova, M. Daneva, A. Herrmann, and R. Wieringa. Agile requirements prioritization: What happens in practice and what is described in literature. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 181–195. Springer, 2011.

[9] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Commun. ACM*, 49:45–47, December 2006.

[10] P. Berander and A. Andrews. Requirements prioritization. In *Engineering and managing software requirements*, pages 69–94. Springer, 2005.

[11] D. Bertolini, A. Novikau, A. Susi, and A. Perini. TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process. Technical report, 2006.

[12] G. Böckle, P. Clements, J. McGregor, D. Muthig, and K. Schmid. A cost model for software product lines. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 310–316. Springer Berlin / Heidelberg, 2004.

[13] B. Boehm. Understanding and controlling software costs. *Journal of Parametrics*, 8(1):32–68, 1988.

[14] B. Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29. ACM, 2006.

[15] B. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches — a survey. *Annals of software engineering*, 10(1-4):177–205, 2000.

[16] B. Boehm, P. Bose, E. Horowitz, and M. J. Lee. Software requirements negotiation and renegotiation aids: A theory-w based spiral approach. In *Software Engineering, 1995. ICSE 1995. 17th International Conference on*, pages 243–243. IEEE, 1995.

[17] B. Boehm et al. *Software engineering economics*, volume 197. Prentice-hall Englewood Cliffs (NJ), 1981.

[18] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605. IEEE Computer Society Press, 1976.

[19] B. W. Boehm, R. Madachy, B. Steece, et al. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.

[20] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[21] J. Brooks, F.P. No silver bullet - essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[22] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[23] C. Burnay, J. Gillain, I. J. Jureta, and S. Faulkner. On the definition of self-service systems. In *Advances in Conceptual Modeling*, pages 107–116. Springer, 2014.

[24] C. Burnay, I. J. Jureta, and S. Faulkner. A Framework for the Operationalization of Monitoring in Business Intelligence Requirements Engineering. *Software and System Modeling (SoSym)*, in press.

[25] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, 27(6):365–389, 2002.

[26] S. Cavaleri and K. Obłój. *Management Systems: A Global Perspective*. Wadsworth, 1993.

[27] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel. Product line analysis: A practical introduction, 1998.

[28] P. P.-S. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[29] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *International Conference on Model Driven Engineering Languages and Systems*, pages 468–483. Springer, 2009.

[30] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.

[31] CISQ Specifications. Automated function points. Technical report, jan 2014. http://www.omg.org/spec/AFP/1.0/.

[32] A. Classen, P. Heymans, and P.-Y. Schobbens. What's in a feature: a requirements engineering perspective. In *Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering*, FASE'08/ETAPS'08, pages 16–30, Berlin, Heidelberg, 2008. Springer-Verlag.

[33] J. Cleland-Huang, G. Zemont, and W. Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 230–239. IEEE, 2004.

[34] P. Clements. On the importance of product line scope. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 102–113. Springer Berlin / Heidelberg, 2002.

[35] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[36] A. Cockburn. Writing effective use cases. *preparation for Addison-Wesley Longman. www. infor. uva. es/˜ mlaguna/is2/materiales/BookDraft1. pdf*, 1999.

[37] M. Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[38] J. J. Cuadrado-Gallego, A. Abran, P. Rodriguez-Soria, and M. A. Lara. An experimental study on the conversion between ifpug and ucp functional size measurement units. *Journal of Zhejiang University SCIENCE C*, 15(3):161–173, 2014.

[39] J. J. Cuadrado-Gallego, L. Buglione, M. J. Domínguez-Alda, M. F. de Sevilla, J. A. G. de Mesa, and O. Demirors. An experimental study on the conversion between ifpug and cosmic functional size measurement units. *Information and Software Technology*, 52(3):347–357, 2010.

[40] A. Dardenne, A. Van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.

[41] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A. M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 179–188. IEEE, 2006.

[42] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[43] M. Denne and J. Cleland-Huang. The incremental funding method: Data-driven software development. *IEEE Softw.*, 21(3):39–47, May 2004.

[44] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, and A. Rifaut. A knowledge representation language for requirements engineering. *Proceedings of the IEEE*, 74(10):1431–1444, 1986.

[45] W. W. Eckerson. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. John Wiley & Sons, May 2008.

[46] J. Eekels and N. F. Roozenburg. A methodological comparison of the structures of scientific research and engineering design: their similarities and differences. *Design Studies*, 12(4):197–203, 1991.

[47] V. Englebert and P. Heymans. Metadone, a flexible metacase to support evolution.

[48] N. Ernst, A. Borgida, and I. Jureta. Finding incremental solutions for evolving requirements. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 15–24. IEEE, 2011.

[49] N. Ernst, A. Borgida, I. Jureta, and J. Mylopoulos. An overview of requirements evolution. In *Evolving Software Systems*, pages 3–32. Springer, 2014.

[50] B. Evelson. The Forrester Wave ™ : Self-Service Business Intelligence Platforms , Q2 2012. Technical report, Forrester, 2012.

[51] F. Ferrucci, C. Gravino, and F. Sarro. Conversion from ifpug fpa to cosmic: within-vs without-company equations. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 293–300. IEEE, 2014.

[52] A. C. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[53] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[54] R. S. Garfinkel and G. L. Nemhauser. *Integer programming*, volume 4. Wiley New York, 1972.

[55] C. Gencel and C. Bideau. Exploring the convertibility between ifpug and cosmic function points: preliminary findings. In *Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2012 Joint Conference of the 22nd International Workshop on*, pages 170–177. IEEE, 2012.

[56] J. Gillain, C. Burnay, I. Jureta, and S. Faulkner. Analyticgraph.com: Toward next generation requirements modeling and reasoning tools. In *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE'16)*. IEEE, 2016.

[57] J. Gillain, S. Faulkner, P. Heymans, I. Jureta, and M. Snoeck. Product portfolio scope optimization based on features and goals. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*, pages 161–170. ACM, 2012.

[58] J. Gillain, S. Faulkner, I. J. Jureta, and M. Snoeck. Using goals and customizable services to improve adaptability of process-based service compositions. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–9. IEEE, 2013.

[59] J. Gillain, I. Jureta, and F. Stéphane. Planning optimal agile releases via requirements optimization. In *Third International Workshop on Artificial Intelligence for Requirements Engineering (AIRE'16)*. Springer, 2016.

[60] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Reasoning with goal models. In *Proc. 21st International Conference on Conceptual Modeling (ER'02)*, pages 167–181, London, UK, 2002. Springer-Verlag.

[61] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. In *Journal on Data Semantics I*, pages 1–20. Springer, 2003.

[62] J. A. Goguen and C. Linde. Techniques for requirements elicitation. *RE*, 93:152–164, 1993.

[63] M. Golfarelli, S. Rizzi, and I. Cella. Beyond data warehousing: what's next in business intelligence? In *Proc. 7th ACM International Workshop on Data warehousing and OLAP*, pages 1–, 2004.

[64] B. Gonzales-Baixauli, J. Prado Leite, and J. Mylopoulos. Visual variability analysis for goal models. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 198–207. IEEE, 2004.

[65] B. González-baixauli, J. C. Sampaio do Prado Leite, and J. Mylopoulos. Visual Variability Analysis for Goal Models. In *Proc.12th IEEE International Requirements Engineering Conference*, pages 198–207, 2004.

[66] O. C. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101. IEEE, 1994.

[67] S. Greenspan, J. Mylopoulos, and A. Borgida. Capturing more world knowledge in the requirements specification. In *Proc. 6th international conference on Software Engineering*, pages 225–234, 1982.

[68] J. Hagelstein. Declarative approach to information systems requirements. *Knowledge-Based Systems*, 1(4):211–220, 1988.

[69] M. Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357. IEEE Computer Society, 2007.

[70] J. C. Harsanyi. Cardinal welfare, individualistic ethics, and interpersonal comparisons of utility. In *Essays on Ethics, Social Behavior, and Scientific Explanation*, pages 6–23. Springer, 1980.

[71] N. C. Haugen. An empirical study of using planning poker for user story estimation. In *AGILE 2006 (AGILE'06)*, pages 9–pp. IEEE, 2006.

[72] M. P. E. Heimdahl and N. G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE transactions on Software Engineering*, 22(6):363–377, 1996.

[73] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(3):231–261, 1996.

[74] A. Helferich, G. Herzwurm, and S. Schockert. Qfd-ppp: Product line portfolio planning using quality function deployment. In H. Obbink and K. Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 162–173. Springer Berlin / Heidelberg, 2005.

[75] A. Herrmann and M. Daneva. Requirements prioritization based on benefit and cost prediction: an agenda for future research. In *2008 16th IEEE International Requirements Engineering Conference*, pages 125–134. IEEE, 2008.

[76] A. M. Hickey and A. M. Davis. A unified model of requirements elicitation. *Journal of Management Information Systems*, 20(4):65–84, 2004.

[77] J. Horkoff, D. Barone, L. Jiang, E. S. Yu, D. Amyot, A. Borgida, and J. Mylopoulos. Strategic business modeling: representation and reasoning. *Software & Systems Modeling*, Oct. 2012.

[78] C. Imhoff and C. White. Self-Service: Empowering Users to Generate Insights. 2011.

[79] I. John and M. Eisenbarth. A decade of scoping: a survey. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 31–40, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[80] C. Jones. Backfiring: Converting lines of code to function points. *Computer*, 28(11):87–88, 1995.

[81] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1):37–60, 2004.

[82] M. Jorgensen, B. Boehm, and S. Rifkin. Software development effort estimation: Formal models or expert judgment? *IEEE software*, 26(2):14–19, 2009.

[83] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on software engineering*, 33(1):33–53, 2007.

[84] H.-W. Jung. Optimizing value and cost in requirements analysis. *IEEE Softw.*, 15(4):74–78, July 1998.

[85] I. Jureta, A. Borgida, N. a. Ernst, and J. Mylopoulos. Techne: towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proc. International Conference on Requirements Engineering*, 2010.

[86] I. Jureta, J. Mylopoulos, and S. Faulkner. Analysis of multi-party agreement in requirements validation. In *2009 17th IEEE International Requirements Engineering Conference*, pages 57–66. IEEE, 2009.

[87] I. J. Jureta and S. Faulkner. Clarifying goal models. In *Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling-Volume 83*, pages 139–144. Australian Computer Society, Inc., 2007.

[88] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens. Clear justification of modeling decisions for goal-oriented requirements engineering. *Requirements Engineering*, 13(2):87–115, 2008.

[89] I. J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. *IEEE International Requirements Engineering Conference*, pages 71–80, 2008.

[90] N. Kano, N. Seraku, F. Takahashi, and S. Tsuji. Attractive quality and must-be quality. 1984.

[91] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14):939–947, 1998.

[92] S. Kelly and J.-P. Tolvanen. Visual domain-specific modeling: Benefits and experiences of using metacase tools. In *International Workshop on Model Engineering, at ECOOP*, volume 2000. Citeseer, 2000.

[93] B. Kitchenham. Counterpoint: The problem with function points. *IEEE software*, 14(2):29, 1997.

[94] M. Kolp and Y. Wautelet. DesCARTES Architect: Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems. *Louvain School of Management, Université catholique de Louvain, Louvain-la-Neuve, Belgium. http://www.i sys.ucl.ac.be/descartes*, 2007.

[95] J. Krogstie, O. I. Lindland, and G. Sindre. Defining quality aspects for conceptual models. In *Information System Concepts*, pages 216–231. Springer, 1995.

[96] K. Lee and K. Kang. Feature dependency analysis for product line component design. In J. Bosch and C. Krueger, editors, *Software Reuse: Methods, Techniques, and Tools*, volume 3107 of *Lecture Notes in Computer Science*, pages 69–85. Springer Berlin / Heidelberg, 2004.

[97] J. C. S. d. P. Leite and P. A. Freeman. Requirements validation through viewpoint resolution. *IEEE transactions on Software Engineering*, 17(12):1253–1269, 1991.

[98] E. Letier and A. Van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 53–62. ACM, 2004.

[99] S. Liaskos, A. Lapouchnian, and Y. Yu. On goal-based variability acquisition and analysis. In *Proc. 14th IEEE International Conference on Requirements Engineering*, pages 79–88, 2006.

[100] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos. Integrating preferences into goal models for requirements engineering. In *2010 18th IEEE International Requirements Engineering Conference*, pages 135–144. IEEE, 2010.

[101] F. J. v. d. Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[102] H. A. Linstone, M. Turoff, et al. *The Delphi method: Techniques and applications*, volume 29. Addison-Wesley Reading, MA, 1975.

[103] G. Low and D. Jeffery. Function points in the estimation and evaluation of the software process. *Software Engineering, IEEE Transactions on*, 16(1):64 –71, jan 1990.

[104] V. Mahnič and T. Hovelja. On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086–2095, 2012.

[105] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. *Requirements Engineering, IEEE International Conference on*, 0:243–253, 2007.

[106] J. Müller. Value-based portfolio optimization for software product lines. In *Proceedings of the 15th International Software Product Line Conference*, SPLC '11, pages 15–24, Washington, DC, USA, 2011. IEEE Computer Society.

[107] J. Mylopoulos and A. Borgida. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362, 1990.

[108] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497, 1992.

[109] M. Nasir. A survey of software estimation techniques and project planning practices. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2006. SNPD 2006. Seventh ACIS International Conference on*, pages 305–310. IEEE, 2006.

[110] D. L. Nazareth and M. A. Rothenberger. Assessing the cost-effectiveness of software reuse: A model for planned reuse. *Journal of Systems and Software*, 73(2):245 – 255, 2004.

[111] S. Negash. Business Intelligence. *Communications of the Association for Information Systems*, 13:177–195, 2004.

[112] R. Neville, A. Sutcliffe, and W. Chang. Optimizing system requirements with genetic algorithms. In *In IEEE World Congress on Computational Intelligence*, pages 495–499, 2003.

[113] M. A. Noor, R. Rabiser, and P. Grünbacher. Agile product line planning: A collaborative approach and a case study. *Journal of Systems and Software*, 81(6):868–882, 2008.

[114] F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *Proceedings of Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003.*, page 308. IEEE, 2003.

[115] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007.

[116] K. Petersen and C. Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of systems and software*, 82(9):1479–1490, 2009.

[117] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[118] J. Poulin. The economics of product line development. *International Journal of Applied Software Technology*, 3:15–28, 1997.

[119] A. Pourshahid, G. Richards, and D. Amyot. Toward a goal-oriented, business intelligence decision-making framework. *E-Technologies: Transformation in a Connected World*, pages 100–115, 2011.

[120] N. A. Qureshi, I. J. Jureta, and A. Perini. Requirements engineering for self-adaptive systems: Core ontology and problem statement. In *Advanced Information Systems Engineering*, pages 33–47. Springer, 2011.

[121] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE transactions on software engineering*, 27(1):58–93, 2001.

[122] A. Ren and C. Yun. Research of software size estimation method. In *Cloud and Service Computing (CSC), 2013 International Conference on*, pages 154–155. IEEE, 2013.

[123] J. Richardson, K. Schlegel, R. L. Sallam, and B. Hostmann. Magic quadrant for business intelligence platforms. *Core research note . . .* , 2008.

[124] C. Rolland, C. Salinesi, and A. Etien. Eliciting gaps in requirements change. *Requirements Engineering*, 9(1):1–15, 2004.

[125] G. Ruhe and M. O. Saliu. The art and science of software release planning. *IEEE software*, 22(6):47–53, 2005.

[126] T. L. Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[127] O. Saliu and G. Ruhe. Supporting software release planning decisions for evolving systems. In *29th Annual IEEE/NASA Software Engineering Workshop*, pages 14–26. IEEE, 2005.

[128] K. Schmid. A comprehensive product line scoping approach and its validation. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 593–603, New York, NY, USA, 2002. ACM.

[129] K. Schmid. An initial model of product line economics. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 198–201. Springer Berlin / Heidelberg, 2002.

[130] K. Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[131] K. Schwaber and J. Sutherland. The scrum guide. *Scrum Alliance*, 2011.

[132] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. *Advanced Information Systems Engineering*, 3084/2004:675–693, 2004.

[133] H. A. Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.

[134] H. A. Simon. *The sciences of the artificial*. MIT press, 1996.

[135] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requirements engineering*, 10(1):34–44, 2005.

[136] I. Sommerville and P. Sawyer. *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[137] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos. (requirement) evolution requirements for adaptive systems. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 155–164. IEEE Press, 2012.

[138] S. Supakkul and L. Chung. The RE-Tools: A multi-notational requirements modeling toolkit. In *Proc 20th IEEE International Conference on Requirements Engineering Conference (RE)*, pages 333–334, 2012.

[139] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques: Research Articles. *Software Practice & Experience*, 35(8):705–754, 2005.

[140] L. J. M. Taborda. Generalized release planning for product line architectures. In R. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 153–155. Springer Berlin / Heidelberg, 2004.

[141] T. T. Tun, T. Trew, M. Jackson, R. Laney, and B. Nuseibeh. Specifying features of an evolving software system. *Software: practice & experience*, 39(11):973, 2009.

[142] M. I. Ullah, G. Ruhe, and V. Garousi. Decision support for moving from a single product to a product portfolio in evolving software systems. *J. Syst. Softw.*, 83(12):2496–2512, Dec. 2010.

[143] R. Valerdi. *The constructive systems engineering cost model (COSYSMO)*. PhD thesis, University of Southern California, 2005.

[144] A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. 5th IEEE International Symposium on Requirements Engineering*, pages 249–262, 2001.

[145] A. Van Lamsweerde. Requirements engineering: from system goals to uml models to software specifications. 2009.

[146] A. Van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE transactions on Software engineering*, 24(11):908–926, 1998.

[147] R. H. Von Alan, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[148] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel. Unifying and extending user story models. In *International Conference on Advanced Information Systems Engineering*, pages 211–225. Springer, 2014.

[149] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, and S. Poelmans. Building a rationale diagram for evaluating user story sets. In *Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on*, pages 1–12. IEEE, 2016.

[150] Y. Wautelet, C. Schinckus, and M. Kolp. Towards knowledge evolution in software engineering: An epistemological approach. In *Systems Approach Applications for Developments in Information Technology*, pages 8–24. IGI Global, 2012.

[151] M. Weber. Keys to Sustainable Self-Service Business Intelligence. *Business Intelligence Journal*, 18:18–24, 2013.

[152] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *2009 17th IEEE International Requirements Engineering Conference*, pages 79–88. IEEE, 2009.

[153] L. A. Wolsey. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2008.

[154] E. S. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 226–235. IEEE, 1997.

[155] P. Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321, 1997.

[156] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM transactions on Software Engineering and Methodology (TOSEM)*, 6(1):1–30, 1997.

[157] Y. Zhang, A. Finkelstein, and M. Harman. Search based requirements optimisation: Existing work and challenges. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 88–94. Springer, 2008.

[158] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM, 2007.

[159] D. Zowghi and C. Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.

[160] D. Zowghi and V. Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software technology*, 45(14):993–1009, 2003.

[161] D. Zowghi and R. Offen. A logical framework for modeling and reasoning about the evolution of requirements. In *Proc. 3rd IEEE International Symposium on Requirements Engineering*, pages 247–257, 1997.